Topográfiai térképek automatikus raszter-vektor konverziója

Doktori értekezés

Szendrei Rudolf

Témavezető: Dr. Elek István

Eötvös Loránd Tudományegyetem
Informatikai Kar

Informatikai Doktori Iskola
Prof. Benczúr András, D.Sc.

Információs rendszerek doktori program
Prof. Benczúr András, D.Sc.

Budapest, 2014.

Tartalomjegyzék

Ábrák jegyzéke

Táblázatok jegyzéke

Köszönetnyilvánítás

Hálás vagyok témavezetőmnek, Elek Istvánnak, hogy egyetemi éveim alatt felkeltette bennem a térinformatika iránti érdeklődést és áldozatos munkájával végigvezetett doktori képzésem rögös útján. Köszönöm neki a sok bátorítást, az őszinte, baráti beszélgetéseket és hogy erőt adott a kutatásomhoz.

Köszönettel tartozom Fekete Istvánnak a bátorításáért, atyai tanácsaiért, a publikálásaim során nyújtott segítségéért és fáradozásaiért. Köszönet illeti Márton Mátyást, aki térinformatikai szaktudásával hozzájárult ismereteim elmélyítéséhez, továbbá köszönöm Fekete Istvánnak, Gercsák Gábornak és László Istvánnak a cikkeimhez nyújtott önzetlen lektorálási és fordítási munkájukat.

Végül, de nem utolsó sorban köszönöm szüleimnek és testvéremnek, hogy végig mögöttem álltak, hogy a legnagyobb lelki támogatást adták munkámhoz, megteremtették a kutatásomhoz szükséges környezetet és mindvégig rendíthetetlenül hittek bennem. Végül köszönöm páromnak a sok-sok türelmet és szeretet, és hogy időről-időre segített a napi teendőkről a kutásomra terelni a gondolataimat.

A „Topográfiai térképek automatikus raszter-vektor konverziója" kutatást kezdetben az IKKK (ELTE Informatikai Kooperációs Kutatási és Oktatási Központ) KMOP-2008-1.1.2 projektje, valamint később az Eötvös Loránd Tudományegyetem TÁMOP-4.2.1/B-09/1/KMR-2010-003 projektje támogatta.

1. Bevezető

A városokat, országokat, kontinenseket összekötő szárazföldi, légi és vízi utak, a közműhálózatok, kommunikációs csatornák és egyéb hálózatok száma ugrásszerűen megnőtt az elmúlt évtizedekben. Bár az utóbbi évtizedben ezen hálózatok helyeit a felújításuk, illetve kiépítésük során már pontosan dokumentálták, a korábbi építésűek nyomvonala sok esetben olyan papír térképeken áll rendelkezésre, melyek beszerzése körülményes, vagy esetleg lehetetlen (akár katonai okokból). Szintén nehézséget jelent, amikor a környezetgazdálkozásban évek, évtizedek hatását vizsgáló kutatók a korábbi mérési eredményeket nem tudják összevetni egy-egy stratégia kidolgozása során (pl. Balaton parti növényzet kontrollálása).

A fenti problémák oka, hogy kezdetben kormányzatilag csak a közmű és kataszteri térképek vektorizálására volt költségvetési keret, illetve a magánszféra csak a GPS rendszerek fejlesztéséhez szükséges mindenkori útvonalak vektorizálását támogatta. Mára ugyan napjaink topográfiai térképek vektorizálásra kerültek, de a régiek vagy anyagi okok miatt, vagy az elévültségük okán már kimaradtak ebből. Sok kutató számára fontos volna azonban, hogy ugyanazon területről, de különböző időben készült térképeket összevethessék és az időbeli változásokat elemezhessék. Régiónkból kitekintve az is látható, hogy például az arab világban, illetve a fejlődő országokban bőven vannak még vektorizálásra váró térképek a hivatalokban.

A vektorizálásra váró topográfiai térképek feldolgozására lehet megoldás az, ha egy olyan rendszert dolgozunk ki, amely képes ezeket automatikusan a raszteres formátumból vektoros modellé konvertálni. Természetesen ekkor is megmarad az emberi interakció szükségessége, hiszen a képeket be kell szkennelni, valamint a térképcsoportokra a szoftvert be kell állítani. Ezen műveletektől eltekintve láthatóvá válik ennek a folyamatnak a vége, ha az egyes térképtárak hajlandóak egy minimális idő ráfordítására és a szoftver üzemeltetésére. Egyetemeken ez valamivel könnyebben válna megvalósíthatóvá a hallgatók segítségével, miközben maguk is sokan tanulhatnának az alkalmazott módszerekről.

Sajnálatos módon nem találtam olyan kész szoftverterméket, amely képes lenne topográfiai térképek maradéktalan vektorizálására. A szoftverek leginkább a fekete-fehér vonalas térképek vektorizálására szorítkoznak (kevés kivételtől eltekintve), és erős megszorításuk, hogy a vektorizálandó bináris kép csak egy térképi réteget tartalmazhat. A fenti okokból adódóan nagy kihívásnak éreztem a topográfiai térképek vektorizálásának automatizálását, mivel ezen térképek egyrészt noha kevés színt, viszont rengeteg színárnyalatot tartalmaznak a nyomdai eljárások, a papír öregedése, a szkenner minősége és egyéb okokból. Másrészt a topográfiai térképek komoly szakértői tudást kívánnak meg értelmezésükkor, hiszen az egymásra rajzolt rétegek egymásból kitakarnak részeket, melyekre esetenként szimbólumok utalnak, máskor ezekre tapasztalt térképolvasó tud csak következtetni. Az általam készített szoftverben a fő célkitűzés az volt, hogy a színes topográfiai térképek logikai rétegeit automatikusan lehessen szétválasztani és a kitakarásból adódó anomáliákat korlátozottan feldolgozni. A rendszer eredményként egy-egy vektoros állományt állít elő a megfelelő pontokból, vonalszegmens-sorozatokból, illetve poligonokból álló rétegekből. A szoftver elkészítése során megismerkedtem a már

meglévő térkép vektorizálási módszerekkel. A rendszer létrehozásakor fő cél volt az, hogy könnyen bővíthető legyen később is, nagy számú térképspecifikus képfeldolgozó algoritmust tartalmazzon, ezek legyenek folyamatba szervezhetőek, a felhasználó pedig könnyen kezelhető felületet használhasson. A képfeldolgozó algoritmusok kiválasztásában segítségemre voltak az IRIS projekt korábbi eredményei és az, hogy kutatásaimat ennek a projektnek a keretében végezhettem. A vektorizálás során alapötletként három feldolgozási szakasz adódott, melyek külön kezelik a topográfiai térképek pontszerű objektumait, hálózati rétegeit, illetve a felületi rétegeket.

Tézisem első részében bemutatom, hogy a piaci szoftverek milyen lehetőségeket kínálnak, illetve milyen cikkek és eredmények adnak támpontot a témában, majd a létrehozott rendszerben alkalmazott megközelítésemet. Ezután tárgyalni fogom az alkalmazott képszűrő eljárásokat, algoritmusokat, valamint a mesterséges intelligencia szükségességét az anomáliák kezelése kapcsán, továbbá rámutatok arra is, hogy a gépi látás mennyiben segítheti a térképi objektumok felismerését. Munkám végén bemutatom a létrehozott rendszert és bemutatok pár, a szoftver segítségével vektorizált térképet is, valamint a hozzájuk kapcsolódó statisztikai eredményeket.

2. Vektorizáló eszközök áttekintése

A papír alapú térképek számítógépes, vektoros modelljének elkészítése iránti igényre válaszul az idők folyamán sokféle hardvereszközt kezdtek el kifejleszteni, melyek közül jó néhány ma is használatban van. Az első vektorizáló eszközök nem nyújtottak semmilyen szoftveres segítséget a felhasználó számára. Ezek olyan hardverek voltak, amelyek segítségével egy már meglévő nyomtatott térképet meg lehetett szerkeszteni a számítógépen. Ezek közül ma is használatos a digitalizáló tábla, toll, valamint a digitalizáló egér, illetve bizonyos termékeknél ezek kombinációja. Egyik lehetséges mód, hogy a térképet a digitalizáló táblára felfektetve a felhasználó sorra megérinti a számára fontos pontokat, melyek a tábla segítségével két dimenziós koordinátákként jelennek meg a számítógépen. Ezután a felhasználó további feladata, hogy ezeket a pontokat a számítógépen összekösse. A pontok segítségével a gépen megfelelő összekötések segítségével létrehozhatók a kívánt vonalak, illetve poligonok. A módszer hátránya, hogy a térkép összes pontjának számítógépre viteléig szükség van az eszközre, melyet más addig nem használhat. Mivel egy ilyen eszköz költséges és helyigényes, valamint tartani kellett a térképnek a géphez viszonyított elmozdulásától a hosszú feldolgozási idő alatt, ezért a lapolvasó eszközök hamar népszerűvé váltak.

A lapolvasó eszközök (szkennerek) gyors elterjedése, alacsony költségük és gyors beolvasási képességük miatt hamar közkedveltté váltak. Segítségükkel a papírtérképeket gyorsan és pontosan lehet raszteres formátumú számítógépes képpé konvertálni. A gyors beolvasás előnye, hogy a térképek mentesülnek a terheléstől, azaz a vektorizálás idejére már ismét a térképtárban lehetnek (elkerülve a rongálódást, mint például pecsétek, szakadás stb.). Ettől a ponttól kezdve, a vektorizálás már szoftveres úton végezhető, melyhez csupán egy személyi számítógépre van szükség és a megfelelő telepített programokra. To-

vábbi pozitívum, hogy a fájlokat helyfüggetlen módon, bármely egyetemen, intézetben, cégnél stb. feldolgozhatják a papírtérkép megléte nélkül.

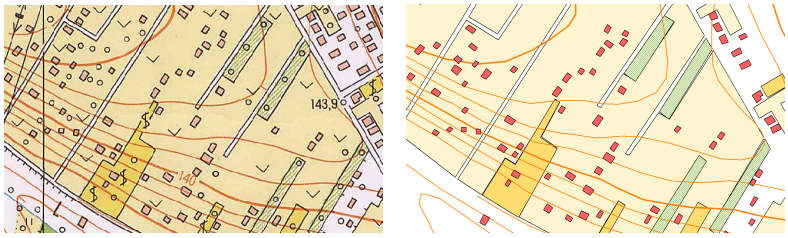
A számítógépes térképszoftvereket két nagy csoportba oszthatjuk aszerint, hogy képes vagy sem önálló képfeldolgozási lépéseket végrehajtani, vagy csupán térképnézegetésre és felhasználói interakció kezelésére alkalmas. A legtöbb térképszoftver alkalmas a raszteres képek megjelenítésére, georeferálására, egyszerűbb műveletek elvégzésére, vektoros megjelenítésre és a vektoros adatokkal való manipulációkra. Az ilyen szoftverek a vektorizálást annyiban támogatják, hogy a felhasználó a megjelenített raszteres térképen a számítógépes egér segítségével körberajzolhatja az épületeket és más poligonokat, valamint az egérrel pontsorozatként adhatja meg a vonalas objektumokat (utak, légvezetékek, folyók stb.). A létrejövő nyers vektoros adatokon már a legtöbb ilyen fajta program képes a poligonok, poliline-ok összevonására, szerkesztésére és tördelésére.

A térképszoftverek másik csoportjába azokat sorolom, amelyek képesek raszteres térképek bizonyos mértékig történő automatikus vektorizálására. A vektorizálandó térképekre természetesen erős megkötések vonatkoznak, így a legtöbb szoftver például csak fekete-fehér vonalrajzból álló térképeket tud feldolgozni. A végeredményként létrejövő vektoros adatok általában valamilyen ismert térképes fájlformátumban kerülnek tárolásra, pl. ESRI Shape formátum, MapInfo TAB formátum (ahol a vektoros adatok a MAP fájlban vannak). A vektorizálás megkezdése előtt általában szükséges a program paramétereinek beállítása (pl. georeferálás), illetve a kész vektoros modellt még további ember általi feldolgozásnak kell alávetni. A következő fejezetekben a két fentebb említett programcsoportba tartozó szoftverekből mutatok be néhányat.

2.1. Manuális vektorizálás

Amint fentebb említettem, a raszter-vektor konverzió szempontjából két csoportba sorolom a térinformatikai szoftvereket aszerint, hogy tisztán emberi munkán alapul, vagy szoftveresen támogatott, esetleg automatikus a vektorizálási feladatok elvégzése. Az első csoportba tartozó szoftverek (pl. MapInfo) támogatják a főbb raszteres képformátumokat, illetve képesek exportálni MAP és SHP vektoros formátumban. Megjelenítő felületük általában támogatja több réteg egymásra rajzolását. A kézi vektorizálás során a kép egy ablakba töltődik be, ahol georeferálás után a rajzeszközök segítségével (pl. 2. ábra mutatja be a MapInfo eszközeit) átrajzolhatjuk a térkép objektumait, melynek egy lehetséges eredménye lentebb látható (1. ábra).

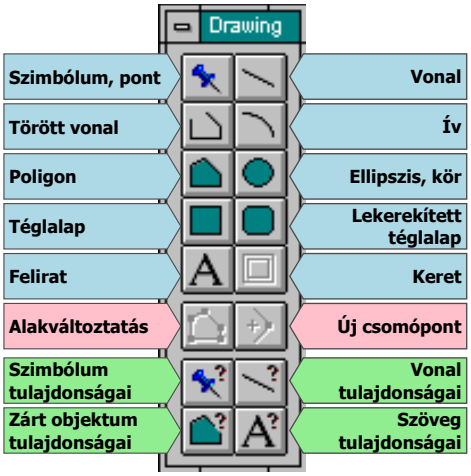
A vektorizálás pár egyszerű lépésben történik. Első lépésként meg kell nyitni a térkép raszteres képfájlját. A raszteres képek megnyitásakor be kell állítanunk, hogy milyen vetületi rendszerben készült a térkép, majd pedig regisztrálnunk kell legalább három darab, nem egy egyenesre eső pontot a georeferáláshoz. A kép a betöltődése után megjelenik a szoftver ablakában. Induláskor általában két réteggel rendelkezünk, az egyik réteg a raszteres, míg a másik réteg a vektoros képet tartalmazza. Utóbbi réteg alkalmas rá, hogy megrajzoljuk rajta az első rétegként vektorizálandó objektumainkat (pl. házak). A vektoros rétegekből többet is létrehozhatunk, így könnyedén szétválogathatjuk a térkép különböző elemeit típusaik szerint már vektorizáláskor külön-külön rétegekre. A mentett



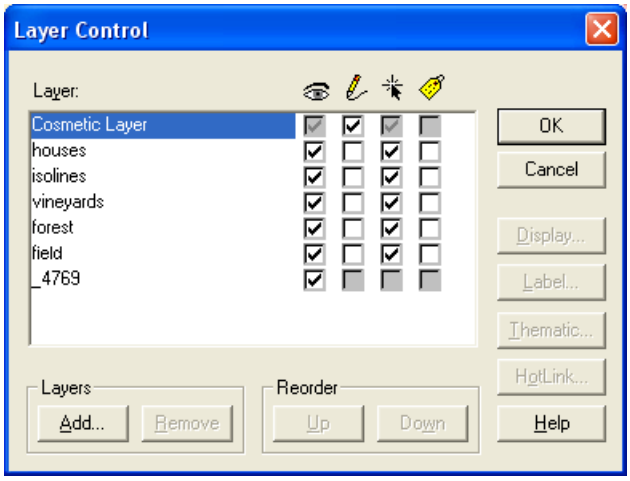
(a) Eredeti térkép (b) Vektorizált térkép

1. ábra. MapInfo használatával végzett kézi vektorizálás eredménye

rétegek automatikusan bekerülnek a rétegek listájába, ahol beállíthatjuk, hogy az adott réteg látható legyen-e éppen, ami igen hasznos lehet az átfedések vektorizálásakor. Ebbe a listába visszatölthetünk természetesen már meglévő rétegeket, félbehagyott vektorizálást stb. Ennek a listának a sorrendjével dönthetjük el azt, hogy az egyes rétegek milyen sorrendben rajzolódjanak egymásra. A mentett rétegek objektumai természetesen később is módosíthatóak vagy törölhetőek (például a MapInfo szoftver réteglistáját a 3. ábra mutatja be).



2. ábra. A MapInfo vektorizálással kapcsolatos eszköztára.



3. ábra. A MapInfo *Layer Control* ablaka, ahol aktuálisan a *Cosmetic Layer* a szerkeszthető réteg. A jelölőnégyzetek oszlopainak jelentése balról-jobbra haladva: láthatóság, szerkeszthetőség, objektumok kijelölhetősége, társított címkék megjelenítése.

2.2. Automatikus vektorizálás és térkép interpretáció

Az automatikus raszter-vektor konverziót illetően célszerű különbséget tenni a különböző típusú térképek között. A fekete-fehér térképek csoportjába általában a kataszteri térképek tartoznak, míg a topográfiai térképek többsége színes. A bináris képeken lényegében vonalak, görbék, feliratok találhatóak, valamint a felületeket jelölő sraffozás, pontozás stb. Az ilyen kataszteri térképek feldolgozása könnyebbnek mondható a színes térképekénél, mivel nem következik be a papíron a színek egymásra nyomtatásánál színkeveredés, illetve az idő múlásával nem változik meg a színük (pl. kifakulás). További könnyebbség, hogy hazánk kataszteri térképeinek rétegei között általában nem, vagy csak csekély mértékben áll fenn kitakarás (pl. telek felett húzódó nagyfeszültségű távvezeték). Közös azonban minden térképtípusban az, hogy a látható képi alakzatok egy térképolvasó ember számára mélyebb összefüggéseket rejtenek magukban (pl. egy azon ívre illesztett vonaldarabok egy szaggatott vonalat jelentenek), ezért szükségszerűen minden térkép vektorizáló rendszernek egyben térkép interpretációs rendszernek is kell lennie.

A térképeket vizsgálva látszik, hogy különböző országok más és más térképkészítési szabályokat alkalmaznak (jelölés, szín stb.). Ezen okból sajnos nem adható olyan általános térkép interpretációs rendszer, amely képes valamennyi térképtípusnak megfelelni, azonban a legtöbb ország a térképeik vektorizálásának idejében igyekezett saját rendszert kifejleszteni. Elmondható, hogy az egyes rendszerek inkább egy-egy térképtípus feldolgozását tűzték ki célul (leginkább a kataszteri térképekét). A következő alfejezetekben bemutatok néhány térkép interpretációs rendszert és röviden ismertetem, hogy milyen módszereket használnak fel működésük során.

2.2.1. FRIMAP – Német topográfiai térképek feldolgozása

A FRIMAP (FRame-based Interpretation of MAPs) német, színes topográfiai térképek (lásd 4. ábra) feldolgozására kidolgozott rendszer [9]. A rendszer a különböző színű területekből első lépésben külön rétegeket hoz létre, majd egy szemantikus háló modellt alkalmaz az interpretáció megvalósítására. A háló a következőket képes elkülöníteni: épület, út, kereszteződés, szintvonal, vízfelület, rét, különböző típusú vegetációk (lombhullató, tűlevelű és vegyes erdők, cserje, magányos fák), cserje. A cikk két érdekes problémára hívja fel a figyelmet a német topográfiai térképekkel kapcsolatban. Egyrészt a rétek nem rendelkeznek konkrét poligonhatárral, mivel ezek felületét csupán pontozás jellemzi (hasonlóan az orosz térképekhez). Másrészt a szintvonalak helyenként megszakadhatnak. Itt a szakadás hosszától függő az, hogy a rendszer képes-e önállóan pótolni azt, mindenesetre képes jelezni a vak végződésűeket ekkor is.

4. ábra. Német kataszteri térkép

2.2.2. MAGELLAN

A MAGELLAN (Map Acquisition of GEographic Labels by Legend ANalysis) rendszer érdekessége hogy öntanító módon próbálja meg felismerni a térképszimbólumokat. Ehhez a térkép alján található jelmagyarázatot használja fel, melyet önállóan lokalizál, illetve szegmentál. A szimbólumok jelmagyarázatból való kinyerése után a szimbólumokhoz a felhasználó által megadott értelmet hozzárendeli, majd pedig tanuló adatokat készít. A tanulási fázis után a térképen végrehajtja a betanult minták felismerését, mely a cikk [19] szerint nagy számú minta alapján mintegy 93%-os pontossággal bír.

2.2.3. MAPINT

A MAPINT (MAP INTerpretation) rendszer, melyet a [12] PhD tézis mutat be, egy általános térképszoftver, amely elsősorban magyar kataszteri térképek feldolgozását támogatja. A program következő fő lépésekből áll:

1. Koordináta transzformáció (a magyar EOV projekciónak megfelelően).
2. Nyers vektorizálás, amely vékonyítás után előallítja a rajz vektoros képét.
3. Felismerések, ahol a térképi objektumokat, valamint helyrajzi számokat megpróbálják automatikusan felismerni, miközben minden egyes lépés után lehetőség van manuális helyesbítésre.

A dolgozatom ezen felül bemutat egy a magyar kataszteri térképeken használt szintvonalrendszert interpretáló módszert is. A módszer segítségével képesek egy külön domborzati réteget létrehozni (szín alapján), valamint ebből a vektorizálás után egy digitális domborzati modellt. A programban megvalósított további felismerő eljárások: szaggatott vonalak felismerése, szimbólumok leválogatása, megírások felismerése és a hozzá kapcsolódó neurális háló alapú karakterfelismerés, valamint kapcsoló jelek, üregek, nullkörök, épületek és telkek felismerése.

2.2.4. MARIS

A MARIS (Map Recognizing Input System) programot japán, nagy léptékű térképek feldolgozására fejlesztették ki. A cikk [20] kitér rá, hogy a térképek feldolgozásának idejét mintegy negyedére sikerült csökkenteniük az emberi munka nagy részének gépi kiváltásával. A program a működése során a felismert objektumokat rétegekbe szervezi, majd ezekből pedig adatbázist épít fel. A térképek feldolgozása négy lépésben történik: előfeldolgozás, vektorizálás, automatikus felismerés, valamint interaktív korrekció.

1. Az előfeldolgozás során a térképet fekete-fehér módban szkennelik be, 16 pixel/mm felbontásban. A képet ezután 70 részre vágják szét, melyek mindegyike 2048 x 1024 pixel felbontású.
2. A vektorizálás során minden pixelt megcimkéznek a rajta átmenő vonal vastagsága szerint, mely kiszámításához vékonyító algoritmust használnak. Ezután gráfot hoznak létre a vékonyítással kapott kép pontjaiból 8-szomszédsági módon. A pontokat csúcsokkal, míg a szomszédságokat éllel reprezentálják. Az elkészült gráfon élritkítást hajtanak végre, majd a csúcsok közül kiválogatják a feature pontokat. Ezen pontok közötti íveket egyenes vonalszegmensekkel közelítik. A kinyert vektoros adatokat így feature pontok, elágazások és szegmensek alkotják.
3. Az automatikus felismerési fázisban három réteget hoznak létre. Külön rétegre kerülnek az épületek és a szintvonalak, illetve egy harmadik rétegre viszik át a vasutakat, utakat és vízfelületeket. Első lépésben az épületeket ismerik fel a gráfban, majd

törlik ezeket onnan. A felismerés során kihasználják azt, hogy az épületek két féle vonalvastagságot tartalmaznak (0.1mm és 0.3mm), valamint a vonalak hossza mindig egy adott értéknél rövidebb. A felismert épületek törlése után a vonalas adatok vektorizálása következik, ahol a vonalak adott vastagságúak és kellően hosszúak. Ennek során vonalkövető algoritmust alkalmaznak.

4. Utolsó lépésként interaktív lehetőséget biztosítanak az adatok kézi pontosítására. Ekkor egér segítségével kijelölhető a módosítandó objektum, ahol meg lehet változtatni a feature pontok helyzetét, törölni lehet őket, illetve létrehozni újakat.

2.2.5. További interpretáló rendszerek

A [28] cikk topográfiai térképek szintvonalainak automatikus interpretációját mutatja be, kitérve arra, hogy a színes képeknél a szintvonalak nehezen azonosíthatóak a színük alapján, az egymásra nyomtatott festékrétegek miatt. A munka színszegmentálásra alapoz, amelyet morfológiai vékonyítás követ. A módszer lényege, hogy a vékonyítás után előálló pontok közül ún. seed pontokat keres c-átlag algoritmussal, majd pedig egy kígyó modell alapján (irányítást figyelembe véve) hízlalja a szintvonalakat, hogy megszüntesse a szakadásokat.

A [16] cikkben bemutatott módszer Bajorországi kataszteri térképek feldolgozására mutat be egy három szintből álló modellt. A modell egyes szintjeihez szemantikus hálókat használ fel.

Az első szintet "szemantikus objektumok"-nak nevezi, és a térkép jelmagyarázatában lévő összes szimbólumot tartalmazza. Második szinten helyezkednek el a grafikák (pont, vonal, görbe, kör, nyíl, sraffozott terület stb.) és szövegek. Ezen a szinten szomszédsági relációk vannak definiálva, ahol a műveletek a vastagságot, pozíciót, méretet és irányítást elemzik. A harmadik szintet "kép gráf"-nak nevezik és számos attribútum-gráfot tartalmaz, melyek mindegyike erősen összefüggő, az objektumok közötti összefüggéseket definiálják (pl. keresztező utaknál az utak szélességére vonatkozó megszorítást). A legfelső szintet "kép"-ként hívják amely maga a szkennelt raszteres kép.

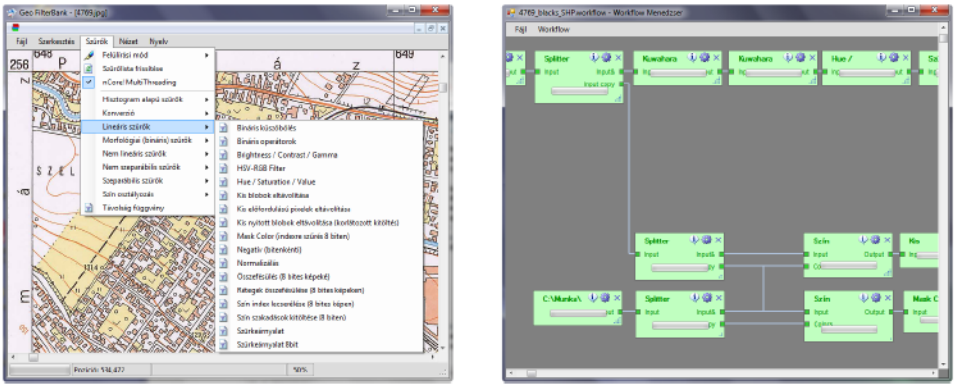
A [12] tézis további, régebben publikált interpretáló rendszereket is ismertet, mint pl. a RoSy rendszert és egy olasz kataszteri térképeket feldolgozót.

3. A GeoFilterBank rendszer ismertetése

Az általam készített GeoFilterBank keretrendszer az IRIS projekt utódjaként, annak kutatási eredményeiből kiindulva, illetve azokat további kutatásokkal kiegészítve került megvalósításra. A korábbi IRIS-ben [6] már korábban folyó kutatások jó kiindulási alapként szolgáltak arra, hogy a raszteres térképek feldolgozásánál milyen digitális képszűrő algoritmusokat célszerű alkalmazni. Ezek nagy része prototípus szinten került megvalósításra, melyek lényege inkább a szemléltetés volt, mint azok egy termelési szoftverbe

A GeoFilterBank rendszer a KMOP-1.1.2-08/1-2008-0002 pályázat támogatásával készült.
Intelligent Raster Interpretation System

történő integrálása. Mivel a rendszer nem rendelkezett grafikus kezelőfelülettel, illetve az egyes funkciók kipróbálása a program újrafordítását kívánta meg, ezért haszna leginkább a megfelelő szűrők és azok esetleges paramétereinek kiválasztásában volt.



5. ábra. A GeoFilterBank program és a Workflow menedzsere.

A fenti kutatási tesztrendszer fejlesztésének átvételekor célomként arra összpontosítottam, hogy létrehozzak egy térképekre optimalizált grafikus szűrőbankot, melynek fő koncepciói a következők voltak:

≤ könnyen kezelhető grafikus felület (lásd 5. ábra),

≤ a szűrők folyamatba rendezhetősége (lásd 5. ábra),

≤ a folyamatok kötegelt módon való futtathatósága,

≤ lazán kapcsolt moduláris felépítés (plugin architektúra),

≤ fejlesztői API a modulokhoz,

≤ API, amely automatikusan párhuzamosítja a szűrők futtatását,

≤ XML alapú kommunikáció a modulok között,

≤ többnyelvűség támogatása.

A program elkészítéséhez a Microsoft Visual Studio 2005 fejlesztőkörnyezetet választottam, ezen belül pedig a Managed C++ programozási nyelvet. Ennek több, előnye is van, többek között a .Net keretrendszerben könnyen lehet grafikus felületet kialakítani, valamint rendelkezésre áll benne az RMI technológia, ami a plugin típusú felépítést nagyságrendekkel megkönnyíti. További előnye, hogy a .Net keretrendszerben a modulok különböző nyelveken írodhatnak, miközben úgyanúgy képesek egymáshoz kapcsolódni.

A GeoFilterBank program egy olyan keretrendszer, ami indításakor gyűjti össze a rendelkezésre álló moduljait, így egy újabb modul létrehozásakor elegendő azt a program könyvtárába másolni ahhoz, hogy használni tudjuk. A modulok az alábbi osztályokba vannak szervezve: import, export, nyelvi és szűrő modulok. A keretrendszer az egyes moduljaival szerializált XML struktúra segítségével, RMI-n keresztül kommunikál, amely azonban eltér más implementációs eljárásoktól oly módon, hogy megengedi a közös memória használatát. A modul-típusok közül a téma szempontjából érdekes szűrő modulokat fogom a következőkben bemutatni, majd pedig azt a kiegészítő nCoreAPI alrendszert [23], amelynek segítségével automatikus módon zajlik a szekvenciálisan programként implementált grafikus szűrők párhuzamosítása. Bár a szűrő modulokat a keretrendszer egységesen kezeli, célszerűségi okokból vizuálisan külön alcsoportokba osztva jelennek meg (ez a modul implementálásakor konfigurálható, hogy melyikben). A szűrő-alcsoportok közül néhány példa: szeparábilis szűrők, morfológiai szűrők stb. (lásd 5. ábra).

4. Digitális szűrők

4.1. Hisztogram alapú szűrők

A hisztogram alapú szűrések esetében a szűrők első lépésként végigfutják a raszteres kép összes pixelét és statisztikát készítenek az egyes intenzitásértékek előfordulásairól, melyet hisztogramnak nevezünk. A képet ezután pixelenként a kapott hisztogram alapján dolgozzák fel. Egy f kép hisztogramja, azaz az egyes intenzitásértékeinek relatív gyakorisága az alábbi módon számítható ki:

$$P(k) = \sum_{i=0,j=0}^{i=n,j=m} \frac{\chi_k(f(i,j))}{n \circ m}, \text{ ahol } \chi_k(p) = \begin{cases} 1 & \text{ha } p = p_k \\ 0 & \text{különben.} \end{cases}$$

4.1.1. Hisztogramkiegyenlítés

A hisztogram kiegyenlítést általában a kifakult képek kontrasztjavítására szokták használni. Látható eredménye csak akkor van, ha a kép legkisebb és legnagyobb intenzitásértéke közül legalább az egyik 0. Ebben az esetben az eljárás a képen előforduló legkisebb és legnagyobb intenzitásérték közötti intervallumot széthúzza a teljes intervallumra, ami 8 bites csatorna esetén a $[0, 255]$ intervallum. Ez azt jelenti, hogy az f kép k csatornájának (i, j) pontjában az $f(i, j, k)$ intenzitásérték a következőképpen módosul, ha a k csatornán az a az előforduló legkisebb, b pedig a legnagyobb intenzitásérték:

$$f^{\circ}(i,j,k) = \frac{(f(i,j,k) \quad a) \circ 255}{b \quad a}.$$

4.1.2. Küszöbölés

A küszöbölő eljárást egy csatornás, azaz szürkeárnyalatos képeken szokták alkalmazni. Általános esetben megadható több küszöbérték, melyek az intenzitásértékeket külön intervallumokba sorolják. Az egyes intervallumokba sorolt pixelek intenzitását ezután az intervallumukhoz rendelt intenzitásértékre állítjuk be. A módszert a gyakorlatban sokszor - és így a programban is - bináris küszöbölésként szokták alkalmazni. Mivel ebben az esetben a pixelek csak két osztályba sorolhatók, amely besorolás nem feltétlen ad jó eredményt.

4.1.3. Otsu-féle küszöbölés

Az Otsu-féle küszöbölés azt a nehézséget hivatott megoldani, hogy a bináris küszöbölésnél minimalizáljuk a pixelek félreosztályozását. Ezt Otsu úgy határozta meg, hogy azt tekintette jó osztályozásnak, ahol a két osztály közötti szórás a lehető legnagyobb. Ehhez először ki kell számolnunk a kép pixeljeinek tapasztalati várható értékét, valamint a szórásnégyzetét. Egy 8 bites szürkeárnyalatos kép esetében ez a következőt jelenti:

$$\mu = \sum_{i=0}^{255} i \circ P(i), \quad \sigma^2 = \sum_{i=0}^{255} (i \quad \mu)^2 \circ P(i).$$

Konkrét t küszöbérték mellett az egyes osztályokon belüli szórás és várható érték:

$$\mu_1(t) = \frac{1}{q_1(t)} \circ \sum_{i=0}^{255} i \circ P(i), \quad \sigma_1^2(t) = \sum_{i=0}^{255} (i \quad \mu_1)^2 \circ P(i), \text{ és}$$

$$\mu_2(t) = \frac{1}{q_2(t)} \circ \sum_{i=0}^{255} i \circ P(i), \quad \sigma_2^2(t) = \sum_{i=0}^{255} (i \quad \mu_2)^2 \circ P(i), \text{ ahol}$$

$$q_1(t) = \sum_{i=0}^{t} P(i), \quad q_2(t) = \sum_{i=t+1}^{255} P(i).$$

Az osztályokon belüli szórás a két osztály szórásának súlyozott összege, vagyis

$$\sigma_w^2(t) = q_1(t) \circ \sigma_1^2(t) + q_2(t) \circ \sigma_2^2(t),$$

míg az osztályok közötti szórás a következő módon definiálható

$$\sigma^2 = \sigma_w^2(t) + \sigma_b^2(t), \text{ vagyis } \sigma_b^2(t) = \sigma^2 \quad \sigma_w^2(t), \text{ kifejtve}$$

$$\sigma_b^2(t) = q_1(t) \circ q_2(t)(\mu_1(t) \quad \mu_2(t))^2 = q_1(t) \circ (1 \quad q_1(t))(\mu_1(t) \quad \mu_2(t))^2.$$

A fentiek ismeretében azt az optimális szórást keressük, ami a legjobban elkülöníti az osztályokat. Látható, hogy az optimumot kétféle módon is megtalálhatjuk: minimalizáljuk az osztályokon belüli szórást, vagy maximalizáljuk a köztük lévőt. Utóbbi megközelítés esetén t függvényében rekurzívan számolható $q_1(t), \mu_1(t), \mu_2(t)$, vagyis

$$q_1(t+1) = q_1(t) + P(t+1), \quad q_1(0) = 0$$

$$\mu_1(t+1) = \frac{q_1(t)\mu_1(t) + (t+1)P(t+1)}{q_1(t+1)}, \quad \mu_1(0) = 0$$

$$\mu_2(t+1) = \frac{\mu \quad q_1(t+1)(\mu_1+1)}{1 \quad q_1(t+1)}.$$

Az implementáció során a következő párhuzamos algoritmus sikerült elkészíteni, mely a processzormagok számával lineárisan gyorsította fel a végrehajtást

1. n processzormag esetén osszuk fel logikailag a képet n közel egyenlő részre, majd minden processzormag egy saját 256 méretű tömbben állítsa elő a hozzátartozó képrészlet hisztogramját.

2. Az elkészült n db hisztogramot összegezzük, majd az így kapott P-n hajtsuk végre az alábbi lépéseket

   (a) Számoljuk ki μ-t és σ-t

   (b) Számoljuk ki $t \neg [0..255]$-re $q_1(t), \mu_1(t), \mu_2(t)$ értékét, illetve ezekből σ_b^2 értékét.

   (c) Válasszuk t_{opt}-nak $argmax_t(\sigma_b^2(t))$-t.

3. Az n processzormag mindegyike végezze el a küszöbölést a t_{opt} küszöbértékkel a saját képrészletén.

4.2. Konverziós szűrők

Az implementált digitális képszűrők többsége hatékonysági okokból és az egyszerű felépítés okán csak bizonyos megszorítások mellett alkalmazhatóak. Például egy színes képre nem tudunk közvetlenül csatornánként Otsu-féle küszöbölést alkalmazni, mivel az csak szürkeárnyalatos képekre hajtható végre. Ezt áthidalhatjuk azonban úgy, ha a képet szétválasszuk csatornáira, ahol minden egyes csatornája egy szürkeárnyalatos kép lesz. Így már végrehajtható a küszöbölés. Amennyiben a küszöbölés eredményei bináris képek lesznek, úgy ezeket ismét 8 bites képekké kell konvertálni, hogy azokat egy 24 bites képpé tudjuk összefűzni. Az ilyen típusú konverziók közül két fajtát tartalmaz a program.

4.2.1. RGB csatornák szétválasztása

Ez az egyszerű szűrő szürkeárnyalatos képeket készít a bemenő színes kép csatornáiból.

4.2.2. UpCast

Az UpCast szűrő feladata egy kisebb bitmélységű képnek a konvertálása egy több bites képpé. Például bináris képet 8 vagy 24 bites képpé, illetve 8 bites képet 24 bites képpé konvertál. Ennek a szűrőnek olyan esetekben van haszna, amikor egy adott bináris térképrétegen elvégzett morfológiai feldolgozás után annak eredményét szeretnénk összefésülni egy tematikus térképpel, amely 8 biten van ábrázolva.

4.3. Lineáris futási idejű szűrők

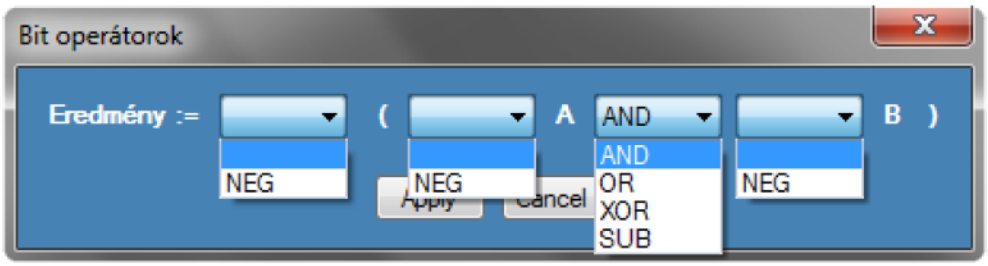
A lineáris futási idejű szűrőket azon szempont szerint soroltam külön csoportba, hogy ezek a képpontokat csak egyszer érintik a feldolgozásuk során. Az ilyen szűrők futási ideje rövidebb, valamint ezeket a szűrőket könnyebb párhuzamos algoritmusok formájában implementálni.

4.3.1. Bit operátorok

A GeoFilterBank rendszerben implementált *Bit operátorok* szűrő (lásd 6. ábra) bemenete két bináris kép. A bemenő képek tetszés szerint első lépésben negálhatóak, majd egy választott operátor kerül végrehajtásra rendre a két kép összes pixelére, ahol az operátor lehet bitenkénti *és, vagy, kizáró vagy, kivonás*. Az így kapott eredménykép szükség szerint negálható. A szűrő működése az alábbi módon írható le röviden:

$r(i,j) = op_4(\ op_1(a(i,j))\ op_3\ op_2(b(i,j))\ )$, ahol

$op_1,\ op_2,\ op_4 \in \{id, \neg\},\ op_3 \in \{\cup, \vee, \wedge, \sqrt{}, XOR\}$.



6. ábra. A GeoFilterBank program Bit operátorok szűrője.

Példa negatív kép előállítására: $a = b,\ op_1 = op_2 = id,\ op_3 = \vee,\ op_4 = \cup$.

4.3.2. Fényerő, kontraszt, gamma

Ez a szűrő egy vagy többcsatornás képeken hajtható végre. Az egyes műveletek, mint fényerő, kontraszt és gamma növelhetőek, illetve csökkenhetőek külön-külön az R, G, B

csatornákon. Fényerő növelése esetén minden pixel intenzitás értéke ugyanakkora értékkel nő, míg csökkentés esetén ugyanakkora mértékben csökken.

$$r(i, j) = \max(\min(f(i, j) + b, 255), 0), \quad \text{ahol} \quad b \neg [\ \ 255, +255].$$

Kontraszt változtatásakor a kép pixeleinek intenzitás értékei közelednek, illetve távolodnak az intenzitás intervallum feléhez viszonyítva, amely 8 bites csatornákat feltételezve a következőképpen írható le:

$$r(i, j) = \max(\min(127 + (f(i, j) \quad\ 127) \circ c, 255), 0), \quad \text{ahol} \quad c \neg [0, 1].$$

A gamma szintén egy széles körben alkalmazott transzformáció, amely az alábbi módon fejezhető ki:

$$r(i, j) = \max(\min(255 \circ \left(\frac{f(i, j)}{255}\right)^{1/g}, 255), 0), \quad \text{ahol} \quad g \neg [0.2, 5].$$

4.3.3. HSB-RGB szűrő

A HSB-RGB szűrő működésének lényege (lásd 7. ábra), hogy egyszerre két különböző színtért használhatunk a kép pontjainak osztályozására. Minden egyes színtér csatornán meghatározhatunk egy intervallumot. Ha egy adott pixel az összes színtérre teljesíti, hogy a megfelelő intervallumon belül esik, akkor elfogadjuk (fehér pixel lesz), egyébként pedig elutasítjuk (fekete pixel lesz). A HSB színtér jellegzetessége miatt megengedettek az úgynevezett ciklikus intervallumok is, ami azt jelenti, hogy a [0.8, 0.1] intervallumot a [0.0, 0.1] ∧ [0.8, 1.0] módon értelmezzük.

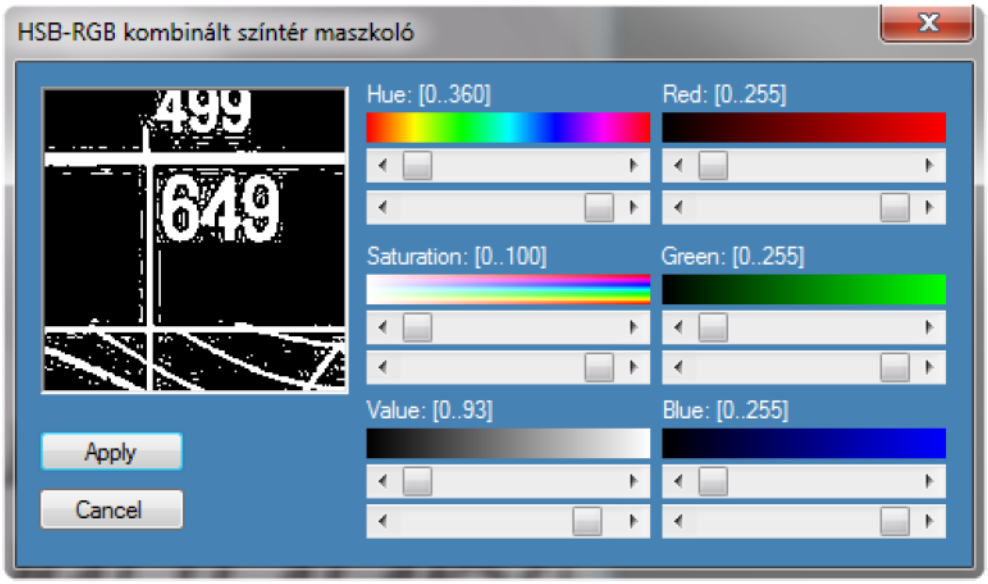
4.3.4. HSB szűrő

A HSB színmodell a színeket három értékkel adja meg: színezet (Hue), telítettség (Saturation), világosság (Brightness). A színezet egy körön 0-359 fokig van ábrázolva, ahol 60 deg a tökéletes piros, 180 deg a tökéletes zöld és 300 deg a kék szín. A telítettség azt adja meg, hogy az adott szín mennyire élénk. Ennek értéke 0 és 1 között lehet, ahol a 0 a színtelenséget jelenti (ilyenkor a színezetnek nincs szerepe), míg 1 értéknél a szín a legtelítettebb. A világosság szintén 0 és 1 közötti érték, ahol a 0 jelenti a feketét, az 1 pedig a fehéret.

A szűrő első lépésként a kép RGB színtérben ábrázolt pixeleit HSB színtérbe konvertálja, majd a paramétereknek megfelelően növeli vagy csökkenti az egyes komponensek értékét. A módosítás után a szűrő visszakonvertálja a kapott eredményt az RGB színtérbe.

Az RGB-HSB konvertálás módja:

1. $max = max(r, g, b)$



7. ábra. A GeoFilterBank program HSB-RGB szűrője.

2. $min = min(r, g, b)$
3. Ha $max = r$ és $g \geq b$, akkor $H = 60 \circ (g \quad b)/(max \quad min)$
4. Ha $max = r$ és $g < b$, akkor $H = 60 \circ (g \quad b)/(max \quad min) + 360$
5. Ha $max = g$, akkor $H = 60 \circ (b \quad r)/(max \quad min) + 120$
6. Ha $max = b$, akkor $H = 60 \circ (r \quad g)/(max \quad min) + 240$
7. Ha $max = 0$, akkor $S = 0$
8. Ha $max > 0$, akkor $S = 1 \quad (min/max)$
9. $B = max$

A HSB-RGB konvertálás módja:

1. $r = g = b = B$
2. Ha $S \neq 0$
3. $sectorpos = hue/60$
4. $sectornum = \lfloor sectorpos \rceil$
5. $fract = sectorpos \quad sectornum$

6. $p = B \circ (1 \quad S)$

7. $q = B \circ (1 \quad (S \circ fract))$

8. $t = B \circ (1 \quad (S \circ (1 \quad fract)))$

9. Ha $sectornum = 0$, akkor $(r, g, b) = (b, t, p)$

10. Ha $sectornum = 1$, akkor $(r, g, b) = (q, b, p)$

11. Ha $sectornum = 2$, akkor $(r, g, b) = (p, b, t)$

12. Ha $sectornum = 3$, akkor $(r, g, b) = (p, q, b)$

13. Ha $sectornum = 4$, akkor $(r, g, b) = (t, p, b)$

14. Ha $sectornum = 5$, akkor $(r, g, b) = (b, p, q)$

15. $(r, g, b) = (r \circ 255, g \circ 255, b \circ 255)$

4.3.5. Kis blobok eltávolítása

A szindetektáló algoritmusok végrehajtása után kapott eredmény egy indexes kép, ahol a pixelekhez nem a színűket, hanem a színük indexét rendeljük. Az indexek 0 és 255 közötti egész értékek lehetnek 8 bites indexelt kép esetén. Az ilyen indexelt képek esetében homogén egybefüggő területnek (blobnak) nevezzük a kép azon pixeleit, melyek azonos indexűek, továbbá létezik őket összekötő, az ő színindexükkel rendelkező pixelekből álló út.

A blobok detektálását a fentiek alapján szokás elárasztásos technikával végezni, amely egy szélességi bejáráson alapul. Ez röviden a következő módon foglalható össze.

1. $blob \uparrow \quad 0$

2. Rendeljük minden pixelhez a $(hamis, 0)$ címkét, amely megmondja, hogy feldolgoztuk-e már az adott pixelt, és ha igen, melyik blobhoz tartozik.

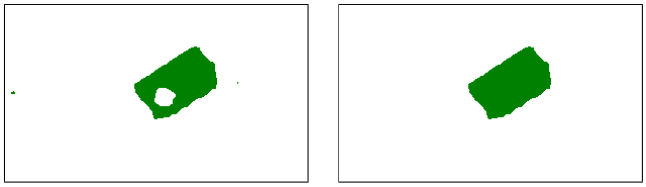
3. A kép pixeleit sorfolytonosan járjuk be. Ha az aktuális pixel címkéjének első komponense hamis, akkor tegyük egy S sorba és növeljük meg *blob* értékét 1-el.

    4. Amíg a sor nem üres, tegyük a következőket:

        5. Vegyük ki az S sorban lévő első pixelt és rendeljük hozzá az $(igaz, blob)$ címkét.
        6. Tegyük be az S sorba a pixelnek a vele azonos színindexű olyan címkéjű szomszédjait, ahol a címke első komponense hamis.

Az algoritmus 6. pontjában említett szomszédokat két féle módon szokásos definiálni. 8-szomszédságnak nevezzük azt, amikor a pixel körüli mind a 8 pixelt szomszédosnak tekintjük. 4-szomszédság esetén csak a pixeltől balra, jobbra, felfele és lefele található szomszédokat vesszük figyelembe.

Látható, hogy az algoritmus lefutása után a *blob* változóban megkapjuk, hogy hány blobot tartalmaz a kép. Az algoritmus könnyen módosítható úgy, hogy a 3. lépésben nulláz egy *méret* számlálót, amit az 5. lépésben eggyel növel. Ennek segítségével ha a 6. lépés után az S sor üres, akkor a *méret* változó az aktuális blob méretét tartalmazza.

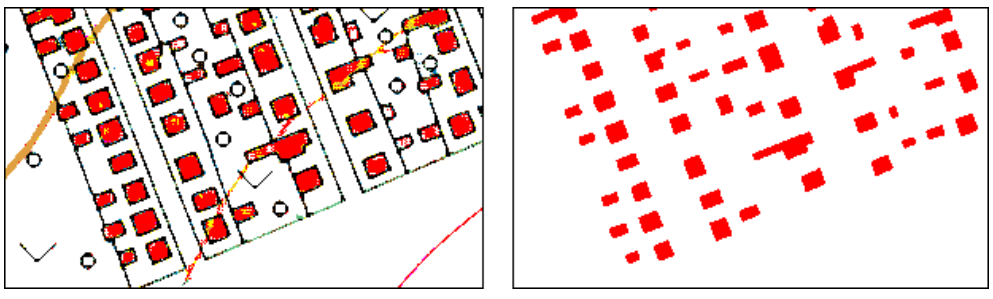


8. ábra. A kis blobok eltávolításának hatása.

A gyakorlatban sokszor van arra szükség, hogy a színdetektálás után kapott területeket úgy osztályozzuk méret alapján, hogy a nem megfelelő méretűeket *töröljük* (lásd 8. ábra). A fent említett módosítások segítségével ez könnyen megtehető, ha az aktuális blob bejárásának végéig megőrízzük pixeleinek helyét. Ezt a nyilvántartást kiürítjük, valahányszor a 3. pontot hajtjuk végre. Ha a detektált blob mérete nem megfelelő, akkor valamennyi pixelének a címkéjét (*igaz*, 0)-ra állítjuk.

4.3.6. Kis nyitott blobok eltávolítása

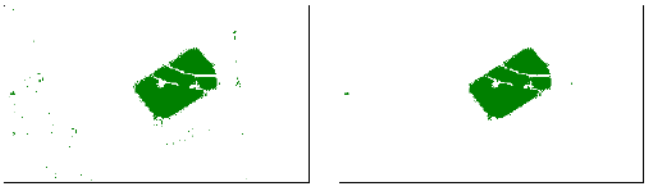
Ez az eljárás az *kis blobok eltávolítása* módszerre épül. Lényege, hogy a szélességi bejáráson alapuló elárasztásos technikát úgy alkalmazza, hogy egy adott színindexű pixelből addig folytatja az elárasztást, amíg vagy el nem éri az elárasztásnál megengedett területméretet, vagy nem ütközik egy előre definiált színindexű határoló pixelbe. Ennek segítségével könnyen lehet olyan eseteket megoldani, ahol bizonyos színfoltokról feltételezzük, hogy egy adott színnel rajzolták őket körbe és területük korlátozott (például házak esetében, lásd 9. ábra).



9. ábra. A GeoFilterBank program Kis nyitott blobok eltávolítása szűrője.

4.3.7. Kis előfordulású pixelek eltávolítása

A szkennelt térképek gyakran tartalmaznak olyan hibákat, amelyek a pixelek helyes osztályozását tovább nehezítik. Egy elvégzett színosztályozás után kapott indexelt képen ezért az osztályozási hibából eredően előfordulhatnak olyan *magányos* pontok, melyek indexe eltér a körülöttük lévő pixelek indexétől. Az ilyen hibákat kiszűrhetjük, ha feltesszük, hogy egy adott pixelnek egy őt körülvevő területet figyelembe véve kell legyen egy minimális előfordulása. Egy $m * n$ méretű képen eszerint egy adott színindexű pontnak a $k * l$ méretű környezetében legalább p-szer kell előfordulnia, különben az indexét egy előre megadott indexre módosítjuk (lásd 10. ábra).



10. ábra. A kis előfordulású pixelek eltávolításának hatása.

Az eljárás egyszerű megközelítésben $m \circ n \circ k \circ l$ képpontot kell, hogy megvizsgáljon, aminél természetesen adható jobb módszer is. Ehhez a [29] cikkben bemutatott *Integral Image* módszer egy módosítását használhatjuk fel. Az eredeti módszer szürkeárnyalatos képeken működik, lényege pedig, hogy létrehozunk egy a térkép felbontásával megegyező, azaz $m * n$ méretű I képet. I adott pontjának értéke az eredeti f kép ezen pontjától a bal felső sarokig terjedő részen lévő képpontok intenzitásértékének összege, vagyis

$$I(i, j) = \sum_{(a,b)=(0,0)}^{(i,j)} f(a, b).$$

A mi esetünkben az új képet úgy definiálhatjuk, hogy az (i, j) koordinátájú pont értéke azt mondja meg, hogy az általunk vizsgált *ind* színindexű pontok hányszor fordulnak elő az imént említett $(i + 1) \circ (j + 1)$ méretű területen, azaz

$$I^{\circ}(i, j) = \sum_{(a,b)=(0,0)}^{(i,j)} \chi(f(a, b) = ind), \text{ ahol } \chi(p) = 1, \text{ ha } p = ind, \text{ egyébként pedig } 0.$$

Ez az új kép $m \circ n$ időben kiszámítható, hiszen

$$\leq I^{\circ}(0, 0) = \chi(f(0, 0) = ind)$$

$$\leq I^{\circ}(i, 0) = I^{\circ}(i \quad 1, 0) + \chi(f(i, 0) = ind)$$

$$\leq I^{\circ}(0, j) = I^{\circ}(0, j \quad 1) + \chi(f(0, j) = ind)$$

$$\leq I^{\circ}(i, j) = I^{\circ}(i \quad 1, j) + I^{\circ}(i, j \quad 1) \quad I^{\circ}(i \quad 1, j \quad 1) + \chi(f(i, j) = ind), \text{ ha } i > 0 \text{ és } j > 0.$$

Ennek alapján tetszőleges $k * l$ méretű téglalapra kiszámíthatjuk konstans időben, hogy hány *ind* színindexű pontot tartalmaz, az alábbi módon:

$$db = I°(i, j) \quad I°(i \quad k, j) \quad I°(i, j \quad l) + I°(i \quad k, j \quad l).$$

Amennyiben $db < p$, akkor az eredeti képen az (i, j) koordinátájú pont indexét lecseréljük egy előre megadott értékre.

4.3.8. Szín maszkolás

Ez a szűrő szintén indexelt képeken alkalmazható, paramétere pedig két színindex, a és b. A képet sorfolytonosan végigfutva a nem a színindexű képpontokat b színindexűre cseréli.

4.3.9. Összefésülés

Az *összefésülés* szűrőnek egy *közös színindex* a paramétere, amelynek segítségével két 8 bites indexelt képből hoz létre egy új képet. Ha a két kép (i, j) pontjainak színindexe megegyezik, vagy az első kép (i, j) pontjának színindexe a közös színindex, akkor a második képen lévő pixel értéke lesz az eredmény, egyébként pedig az első képen lévő pixel értéke. Ennek az a jelentése, hogy a második kép csak azon pontjaiban rajzolódhat az alatta lévő első képre, amely pontokban az alsó kép közös színindexű pixelt tartalmaz.

4.3.10. Rétegek összefésülése

Ez a szűrő szintén két darab 8 bites indexelt képet fésül össze (alap B és extra E réteget), egy R indexelt képpé. Három paramétere van: b alap színindex, e extra színindex és egy k küszöbérték. A két kép képpontjaira a következőket kell végrehajtani:

1. $R(i, j) \uparrow \quad B(i, j)$
2. Ha $B(i, j) \neq e$ és $E(i, j) = e$, akkor
   (a) Számoljuk meg $B(i, j)$ hány szomszédja e színindexű, legyen ez d.
   (b) Ha $d <= k$, akkor legyen $R(i, j) = b$.

Az algoritmus 1. lépése az alapréteget átmásolja a célképre. A 2. lépés eldönti, hogy az alaprétegen extra szín található-e az adott pontban. Amennyiben igen, és fennáll, hogy az extra rétegen ebben a pontban extra szín található, akkor meg kell vizsgálni az alapréteg pontjának szomszédait is. A szomszédok esetében megszámoljuk, hogy hánynak van extra színindexe. Amennyiben kevesebb, mint k darab ilyen szomszédot találtunk, úgy a célkép adott pontjának értékét egy előre meghatározott b színindexre cseréljük.

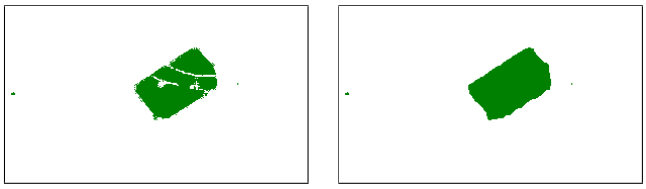
Az eljárás röviden tehát annyit tesz, hogy az első kép azon nem extra színindexű pontjait cseréli le alap színindexűre, amelyeknek nincs elegendő számú extra színindexű szomszédjuk és a felettük lévő második kép pontja extra színindexű.

4.3.11. Szín index lecserélése

A szín index lecserélése egy egyszerű művelet, melyben a kép összes a szín indexű pontját b indexűre cseréljük. Ez akkor bizonyul hasznosnak, amikor egy színdetektálás során egy szimbólum több szín osztályba is esik a különböző színű részei miatt. Ilyenkor $a = b$ választással az a és b osztályokat összevonhatjuk egy közös index alá.

4.3.12. Szín szakadások kitöltése

A topográfiai térképeknél gyakori jelenség, hogy a felületeken áthaladó vonalas objektumok, szimbólumok és feliratok miatt a felületi réteg vizuálisan megszakad. Kisebb szimbólumok esetén ez *lukakat* jelent, amik blobokként kezelhetők, így a korábban tárgyalt szűrők segítségével ezek eltávolíthatóak. A vonalas objektumok miatt létrejött takarás ennél bonyolultabb eset, mivel ezek területére nem adható általános becslés. Ehelyett egy olyan módszert sikerült kidolgoznom, amely a következő paraméterekkel szabályozható: a felületi szín c (amely megmondja, hogy melyik felületi szín index szakadásait kívánjuk megszüntetni), illetve a szakadás mértéke d (hány pixel távolságban lehetnek az ugyanahhoz a felülethez tartozó pontok), valamint az iterációk száma i, (azaz hányszor ismételjük a szűrő végrehajtását).



11. ábra. A szín szakadások kitöltésének hatása a képen.

Az eljárás i-szer fut le az alábbi módon. A képet átmásolja egy átmeneti képre, majd végigfut az eredeti kép pixelein sorfolytonosan. Az egyes iterációk végeztével az eljárás az átmeneti kép tartalmát visszamásolja az eredeti képre. Ha az adott képpont nem c szín indexű, akkor a pontból szimultán keresést indít két ellentétes irányban d távolságon belül. Amennyiben mindkét irányban talál c szín indexszel rendelkező pontot, úgy a vizsgált pontnak megfelelő koordinátájú pontot c értékűre módosítja az átmeneti képen (lásd 11. ábra). Az eljárás az ellentétes irányban végzett keresést további 3 irányban kísérli meg hasonlóképpen, ha nem talál az ellentétes irányokban c indexű pontokat. Az egyes irányok a következők, melyekkel ellentétes irányokat szimultán vizsgálja az algoritmus: $(1, \ 1), (1, 0), (1, 1), (0, 1)$.

4.3.13. Szürkeárnyalat

Bár a 24 bites *true color* képeket könnyen átalakíthatjuk szürkeárnyalatossá a más korábban ismertetett HSB szűrő segítségével, az RGB-HSB és HSB-RGB konverzió műveletigényes volta miatt célszerű volt erre a feladatra egy külön szűrőt készíteni. A hagyományos szürkeárnyalat kiszámítás helyett, amely az $L = (R + G + B)/3$ átlagolást

használja, a térképészetben és a természetképeknél jobban használható eljárás került implementálásra: $L = 0.299 \circ R + 0.587 \circ G + 0.114 \circ B$, amely jobban modellezi az emberi szemnek a különböző hullámhosszakra való érzékenységét.

4.4. Konvolúciós szűrők

A konvolúciós szűrőket gyakran lineáris szűrőkként is szokták emlegetni, működésüket tekintve pedig egy súlyokat tartalmazó w mátrixot használnak fel. Ezt a $(k+1)*(k+1)$ méretű mátrixot kernelnek nevezik, és egy f kép $f(x,y)$ pixelének intenzitásértéke úgy számolható ki, hogy a környezetében lévő intenzitásértékeket a kernelben lévő súlyokkal átlagoljuk:

$$\sum_{i=x\ k,\ j=y\ k}^{i=x+k,\ j=y+k} f(i,j) \circ w(i \quad x+k, j \quad y+k).$$

4.5. Szeparábilis szűrők

A szeparábilis szűrők w kernelmátrixa speciális tulajdonságú, mivel felbontható egy oszlop- és egy sorvektor szorzatára az alábbi módon:

$$w(i,j) = u(i) \circ v(j).$$

Ennek segítségével a szűrő eredménye úgy is kiszámolható, hogy előbb kiszámoljuk a kép konvolúcióját u-val, majd ennek eredményét konvolváljuk v-vel, azaz

$$f^{\circledast}(x,y) = \sum_{i=x\ k}^{i=x+k} f(i,y) \circ u(i \quad x+k), \text{ majd pedig}$$

$$g(x,y) = \sum_{j=y\ k}^{j=y+k} f^{\circledast}(x,j) \circ v(j \quad y+k).$$

A konvolúció tehát szeparábilis esetben $(2k+1) \circ (2k+1)$ művelet helyett elvégezhető $2(2k+1)$ művelettel. Ennek feltétele, hogy rendelkezzünk $O(n \circ m)$ segédmemóriával, amennyiben a kép $n * m$ méretű.

4.5.1. Átlagszűrő

Az átlagszűrő, vagy más néven box szűrő egy olyan speciális kernelű szűrő, amelynek mátrixában minden érték ugyanakkora. A szűrő képsimításra alkalmazható, azonban önmagában ritkán használatos.

4.5.2. Gauss szűrő

Ennek a szűrőnek a kernele a két dimenziós Gauss eloszlás értékeit tartalmazza. A Gauss eloszlás

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{(x^2+y^2)}{2\sigma^2}}$$

képletét vizsgálva látható, hogy ez egy 0 várható értékű, σ szórású Gauss haranggörbét ad, amely az x és y tengelyre nézve szimmetrikus. Ez alapján a kernele felbontható két vektor szorzatára, melynek értékei az alábbi módon számíthatóak ki:

$$\frac{1}{2\pi\sigma^2} e^{\frac{(x^2+y^2)}{2\sigma^2}} = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x^2)}{2\sigma^2}} \circ \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(y^2)}{2\sigma^2}}.$$

Ez a szűrő szintén simító szűrőként alkalmazható. A gyakorlatban alkalmazzák a szűrő speciális változatát a művelet gyorsítására úgy, hogy a haranggörbe értékeit 2 hatványokkal közelítik (ekkor bit eltolás használható szorzás helyett). A simítás mértéke a szórás módosításával állítható be. Figyelembe kell azonban venni, hogy a szórás növelésével egy időben a kernel méretét is megfelelően kell növelni.

A szűrőt leginkább más algoritmusokkal együtt használják, mint pl. a Canny éldetektorban, vagy a Gauss-piramis készítésénél.

4.6. Nem szeparábilis szűrők

Ebbe a csoportba azok a konvolúciós szűrők tartoznak, amelyek mátrixa nem írható fel két vektor szorzataként.

4.6.1. Éldetektorok

A programcsomagban megvalósított éldetektorok között szerepel többek között a Prewitt és Sobel operátor. Ezek a gradiens szűrők csoportjába tartoznak. A gradiens szűrők a képet felületként értelmezik, és az adott pontban vett deriváltak x és y irányú gradiensét közelítik különbségi hányadossal. Ennek az az eredménye, hogy a homogén felületeken 0 közeli értéket adnak, míg a nagy intenzitáskülönbséggel rendelkező területeken pedig nagy értékeket. Egy általános gradiens szűrő kernele a következőképpen írható le (lásd 12. ábra).

$G_x =$

-1	0	1
2-p	0	p-2
-1	0	1

$G_y =$

-1	2-p	-1
0	0	0
1	p-2	1

12. ábra. Az x és y irányú gradienseket közelítő általános szűrő kernelek.

p értéke szabadon megválasztható, de a gyakorlatban a Prewitt szűrőt adó $p = 2$, a Sobel szűrőhöz tartozó $p = 3$, valamint az izotropikus operátornál alkalmazott $p = 2 + \sqrt{2}$ értékek a leginkább használatosak. A Prewitt és Sobel szűrő eredménye látható többek között az alábbi ábrán (13. ábra).

(a) Eredeti kép (b) Laplace-I. (c) Laplace-II. (d) Sobel (e) Prewitt

13. ábra. Éldetektorok hatása a képen.

Az éldetektorok közül szintén hasznosnak bizonyult a Laplace szűrő. A Laplace operátor definíciója a következő:

$$L(f(x,y)) = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y^2}.$$

Mivel a raszteres képek diszkrét mintavételezésből származnak, ezért érdemes a Laplace operátort diszkrét esetben vizsgálni. A pontbeli második deriváltat a kép diszkrét pontjaiban vett értékekkel (azaz a szomszédos pixelek értékével) számolhatjuk ekkor:

$$f^{\infty}(x,y) = f(x \quad 1,y) + f(x+1,y) + f(x,y+1) + f(x,y \quad 1) \quad 4 \circ f(x,y).$$

Ez alapján a következő kernel használható a konvolúcióhoz (lásd 14. ábra).

0	1	0
1	-4	1
0	1	0

14. ábra. A Laplace szűrő kernelje.

A fenti Laplace operátornak van egy elterjedt másik változata is, ahol a kernel mátrix középpontjában a -8 érték áll, az összes többi helyen pedig az 1-es érték. A programban az előbbi szűrő a Laplace-I, míg utóbbi a Laplace-II operátor nevet kapta.

4.6.2. Kép élesítése

A képek feldolgozásakor szükséges lehet, hogy a képeket élesítsük, melynek jelentése, hogy a pixelek értékét távolítjuk a környező pixelek értékétől. Ezt megtehetjük természetesen a fentebb ismertetett szűrők segítségével. Vegyük a képet, majd adjuk hozzá a kép és az átlagolt kép különbségét, vagyis

$$f^{\circ}(x,y) = 2 \circ f(x,y) \quad f_{\text{sima}}(x,y).$$

Az f_{sima} kép az eredeti kép simított változatát jelenti. A kép simítására a korábban említett szűrők közül választhatunk például átlag vagy Gauss szűrőt.

4.7. Nem lineáris szűrők

Ezen szűrőcsoport onnan kapta elnevezését, hogy a kép feldolgozásakor az egyes pixelek értékét noha a szomszédos pixelek intenzitásának felhasználásával számítjuk ki, de nem a lineáris kombinációjukként. Ebből adódóan nem adható meg kernel mátrix, helyette inkább ablakról beszélhetünk, amely alá eső terület határozza meg a pixel új értékét. Az új pixelértékeket különböző módon számíthatjuk ki, mint például a környezet szórásának ismeretében, vagy a környezet pixelértékeinek sorba rendezett értékei alapján (lásd Medián szűrő).

Ezen szűrőkről általánosságban elmondható, hogy futási idejük nem lineáris (noha elnevezésük a korábban említett tulajdonságukból fakad). Ennek az az oka, hogy az új pixelértékek kiszámítására fordított idő a pixeleken végigvitt ablakmérettől függ. Bizonyos adaptív szűrők esetén ez az ablakméret a feldolgozás során a terület tulajdonságától függően változhat.

4.7.1. Bináris mintaillesztés

Bináris mintaillesztésre azon esetekben lehet szükség, amikor egy bináris kép részleteinek egy előre megadott bináris képhez vett hasonlóságát akarjuk vizsgálni. Ilyen megoldást alkalmazhatunk például a pontszerű szimbólumok detektálásakor is. Ilyenkor feltesszük azt, hogy az 1-es érték a háttérhez tartozó fehér pixel értéke, míg a 0-ás érték az objektumok fekete színét reprezentálja.

A módszer egy $m * n$ méretű f képen keresi a $k * l$ méretű g kép előfordulásait úgy, hogy eredményként egy, az f kép méreteivel megegyező h képet hoz létre. g képet mintának hívjuk, $h(x,y)$ értéke pedig 1 vagy 0 lehet, melyet a következő módon határozhatunk meg.

1. $h(x,y) \uparrow \quad 0.$
2. $fekete \uparrow$ (g mintában lévő 0-ák száma (a minta fekete pixeleinek száma)).
3. Képzeletben toljuk el g-t az $(x \quad k/2, y \quad l/2)$ pontba f-en.

4. $s_{=} \uparrow \sum_{i=0,j=0}^{i=k\ 1,j=l\ 1} 1 \quad \backslash g(i,j) \quad f(x+i \quad k/2, y+i \quad l/2) \backslash$

5. $s_f \uparrow \sum_{i=0,j=0}^{i=k\ 1,j=l\ 1} \backslash g(i,j) \quad f(x+i \quad k/2, y+i \quad l/2) \backslash \circ f(x+i \quad k/2, y+i \quad l/2).$

6. $s_g \uparrow \ k \circ l \quad s_f \quad s_{=}.$

7. $max \uparrow \ 3 \circ s_f + s_g$

8. Ha $\frac{max \circ 100}{k \circ l} < thres$, akkor $h(x,y) \uparrow \ 1.$

A fenti pszeudokódban három különböző tulajdonságot vizsgálunk. $s_{=}$-ben megszámoljuk a minta és a képrészlet közötti egyezéseket, míg az eltéréseknek két részesetét vizsgáljuk. s_f-ben megszámoljuk, hány pontban tér el f képrészlete a mintától, ahol a mintában fekete pixel található (azaz 0-ás érték). s_g hasonlóan definiálható, ez azt számolja meg, hány pontban tér el a minta az f képrészletétől, ahol f fekete pixelt tartalmaz.

Az algoritmus aszimmetrikus módon *bünteti* az eltéréseket, ahogy az *max* értékének definiálásából látható. Eszerint kevésbé megengetett az, ha az f képrészletében a mintának megfelelő ábra sérült, mintha a képrészlet a minta szempontjából háttérként kezelt területen objektumpixeleket tartalmaz. A minta előfordulását akkor fogadja el az algoritmus ha az eltérés egy megengedett *thres* küszöbérték alatt van.

A megoldás látszólag időigényes, hiszen $k \circ l$ az, ahányszor az egyes pixeleket érintjük. A gyakorlatban a futási idő optimalizálására több módszer is alkalmas:

≤ *fekete* értékét elegendő egyszer kiszámítani az egész kép feldolgozása előtt.

≤ A bináris képekben 8 pixel egy bájton van ábrázolva, így egyszerre 8 pixelt tudunk feldolgozni.

≤ A minta képhez képesti eltolása megoldható egyetlen *biteltolás* művelettel, ami után a kép kellő részének kimaszkolása elérhető egyetlen *bitenkénti és*-sel.

≤ Az s egy bájtos változóban ábrázolt 8 pixelből az 1-esek megszámlálása rendkívül hatékonyan végezhető el 32 bites környezetben:

1. v = s - ((s >> 1) & 0x55555555)
2. v = (v & 0x33333333) + ((v >> 2) & 0x33333333)
3. e = ((v + (v >> 4) & 0xF0F0F0F) * 0x1010101) >> 24

≤ Pozitív találat esetén a találat környezete kimaszkolható, így azt nem kell már megvizsgálnunk.

≤ A szűrőt párhuzamosítva m processzormag esetén a futási időt tovább csökkenthetjük m-edére.

Az algoritmus futási ideje a fentiek alapján N képpontból álló f kép esetén $\theta(N) = N \circ \frac{k \circ l}{8 \circ m}$. Emellett azonban érdekes megjegyezni, hogy az algoritmust sikerült úgy implementálni, hogy az 32 pixel (azaz 32 bit) feldolgozásához 45 műveletet végez. Ez végeredményben azt jelenti, hogy a párhuzamosítás segítségével egy 4 processzormagos gépen egy pixel feldolgozásához $\ll 0.3$ művelet elvégzésére van szükség. A szűrő végrehajtási ideje 300 pixeles mintákat alapul véve az $5 * 5$ méretű kernellel rendelkező nem-szeparábilis szűrők futási idejéhez volt mérhető.

4.7.2. Medián szűrő

A medián szűrő egy speciális rank szűrő. A rank szűrők első lépésként növekvő sorba rendezik a pixel és környezete intenzitás értékeit. A medián szűrő ebből a sorból mindig a középső értéket választja a pixel új intenzitás értékének. Ebből következik az, hogy a medián szűrőhoz nem feltétlen van szükség rendezésre, hiszen a k-adik elem kiválasztása módszer is elegendő a feladat megoldásához, ráadásul sokkal hatékonyabb is. A $(2k + 1) * (2k + 1)$ ablakméret mellett így a $\theta(N) = N \circ 2log(2k + 1) \circ (2k + 1)^2$ futási idő, ami a rendezésből adódó $2log(2k+1)$ taggal szorzódik, lecsökkenthető $\theta(N) = N \circ (2k+1)^2$ időre. Bár a medián szűrő jól alkalmazható a *salt and pepper* (só, bors) hibák eltűntetésére, sajnos az éleket nem őrzi meg. Működéséből következik továbbá, hogy a lokális hibákon túl eltávolítja azokat a pontokat, vonalakat, melyek k-nál vékonyabbak.

4.7.3. Konzervatív simítás

A konzervatív simítás szűrőt szokás élmegörző medián szűrőként is nevezni. A szűrőt szürkeárnyalatos képeken alkalmazzuk a következő módon. Vesszük a pont szomszédainak intenzitás értékeit és megkeressük közülük a minimálisat és a maximálisat. Amennyiben a képpont intenzitásértéke ezen $[min, max]$ intervallumon kívül esik, úgy a pont új értéke a hozzá közelebb eső intervallum végpont érték lesz, azaz

$$f(x, y) < min \text{ , } \quad f(x, y) \uparrow \quad min$$

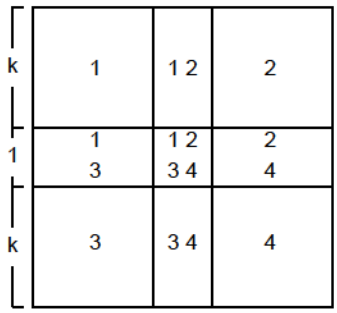
$$f(x, y) > max \text{ , } \quad f(x, y) \uparrow \quad max.$$

A k paraméterű szűrő futási idejét a használt $(2k + 1) * (2k + 1)$ ablakméret határozza meg, így $\theta(N) = N \circ (2k + 1)^2$. A minimum és maximum érték a szomszédok közül szimultán kiválasztható, miközben a szomszédok értékeit lekérdezzük. A szűrő hasznos tulajdonsága, hogy eltávolítja a *só, bors* hibaként nevezett zajokat, miközben az objektum sarkát nem kerekíti le.

4.7.4. Kuwahara

Ez a szűrő szintén lokális zajok eltávolítására használható, és a konzervatív simítás továbbfejlesztett változatának tekinthető. k paraméterének megadása után az ablakmérete

$(2k + 1) * (2k + 1)$, ahol a vizsgált pont az ablak középső eleme. Az ablakban lévő pontok közül kiszámítjuk az ablak bal-felső, jobb-felső, bal-alsó és jobb-alsó sarokban lévő $(k + 1) * (k + 1)$ méretű négyzetekbe eső pontok tapasztalati szórását és átlagát (lásd 15. ábra). A ponthoz ezután annak a négyzetnek az átlagértéket rendeljük, amelyik négyzetnek minimális a szórása.



15. ábra. A Kuwahara szűrő futásakor vizsgált négy négyzet alapú terület. Itt az i-edik négyzet azt a területet fedi le, amelyen az i index szerepel. A négy négyzet metszete az a pont, amelynek új értékét szeretnénk meghatározni.

Az eljárás élmegőrző hatású, ami abból következik, hogy ha egy pont egy él mellett helyezkedik el, akkor növelnénk a szórást, ha az él másik oldaláról választanánk meg a pixelek átlagát (hiszen azon pixeleknek a ponthoz viszonyítva nagy lesz az intenzitás különbségük).

4.7.5. Gamma szűrő

A Gamma szűrőt elsődlegesen radar képek szűrésére használatos, mivel a nagy frekvenciás zajokat távolítja el úgy, hogy közben megőrzi a nagy frekvenciás jellegzetes részeket (vagyis az éleket) a képen. A szűrőt először Kuan mutatta be és működéséhez a kép valószínűségi sűrűségfüggvényének a-priori ismeretére van szükség. Emellett feltevés, hogy a kép hibája Gauss eloszlást mutat. Lopes később módosította a szűrőt. Ő is fenntartotta, hogy a hiba Gauss eloszlású, de két küszöbértéket vezetett be módszerében. Az általam használt szűrő a Lopes féle változat [15].

A szűrő alkalmazásánál használt $(2k + 1) * (2k + 1)$ ablakméret mellett k értékét 1 és 5 közé szokták választani, mivel $k = 1$ esetén a szűrő hatása elenyésző, míg $k = 5$ esetén már erősen elmossa a nagyobb részleteket is. A szűrő további két paramétere *Look* és *Var*. A *Look* paraméter azt fejezi ki, hogy a radarkép hányszor képződött le (hány radarpásztázás eredménye lett a képen egymásra rajzolva), míg *Var* a zaj szórását adja meg. A szűrő ezek alapján egy szürkeárnyalatos kép egy pontjához a következő intenzitásértéket rendeli:

$$R = \begin{cases} I & \text{ha } C_i \geq C_u, \\ \dfrac{B \circ I + \overline{D}}{2 \circ \alpha} & \text{ha } C_u < C_i < C_{max}, \\ CP & \text{ha } C_{max} \geq C_i. \end{cases}$$

$\alpha, B, C_i, C_u, C_{max}, CP, D$ és I értékek a következő módon számolhatóak ki:

$$\begin{aligned} Var &= \text{A hiba szórásának mértéke a szűrő ablakban} \\ CP &= \text{A középső pixel intenzitásértéke} \\ I &= \text{Az ablakban lévő pixelek intenzitásértékének átlaga} \\ C_u &= \frac{1}{\overline{Look}} \\ C_i &= \frac{\overline{VAR}}{I} \\ C_{max} &= \overline{2} \circ C_u \\ \alpha &= \frac{1 + C_u^2}{C_i^2 \quad C_u^2} \\ B &= \alpha \quad Look \quad 1 \\ D &= I^2 \circ B^2 + 4 \circ \alpha \circ Look \circ I \circ CP \end{aligned}$$

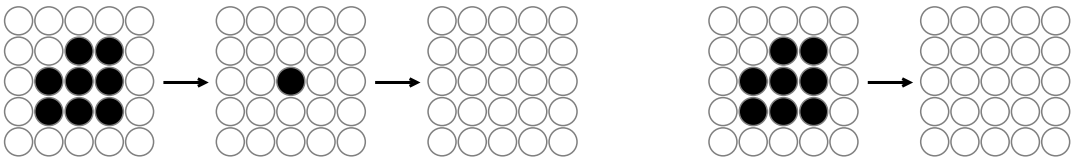
Noha a Gamma szűrőt radarképek feldolgozására tervezték, a topografikus térképek előfeldolgozásánál is kíválóan használható az említett képzajok eltávolítására. A szűrőt szürkeárnyalatos képekről könnyen ki lehet terjeszteni színes képekre, ilyenkor a színcsatornákat egyenként kell feldolgozni. Ennek a kiterjesztésnek egy érdekes esete, amikor az RGB-től eltérő színtérben alkalmazzuk a szűrőt. Például YCbCr színmodell esetén a JPEG és tömörített TIFF formátumban tárolt képek krominancia csatornáin fellépő tömörítési zajokat lehet csökkenteni.

4.8. Morfológiai és bináris szűrők

Ezen szűrőcsoporton belül több szűrő került megvalósításra, melyek a térképfeldolgozás szempontjából hasznosak a vonalas adatok kinyerésében. Az interpretáció során a korábbi lépések eredményeként létrejött bináris képekből indulunk ki. Feltesszük, hogy az ilyen képeken az 1-es érték jelöli, hogy a pixel objektumhoz tartozik, a 0 érték pedig, hogy háttérhez. A következőkben ismertetett morfológiai szűrők alkalmazásával azt szeretnénk elérni, hogy a több pixel vastagságú vonalas adatokat egy pixel vastagságúra vékonyítsuk, miközben topológiájukat megtartjuk.

4.8.1. Erózió, kiterjedés

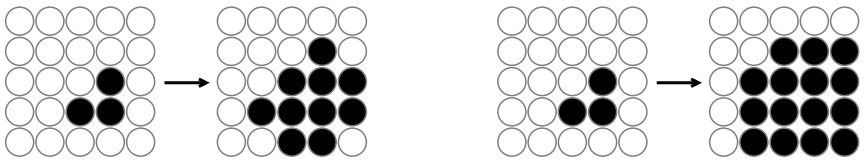
Az erózió művelet a bináris kép feldolgozásakor eltávolítja a nem *belső* pixeleket. A belső pixelek definiálása történhet 4-szomszédság, illetve 8-szomszédság alapján. 4-szomszédság esetén belső pixelnek tekintjük azt az 1-es értékű (fekete) pixelt, amely alatt, felett és tőle jobbra, illetve balra is 1-es értékű pixel található. A 8-szomszédság esetében az 1-es értékű belső pixeltől megköveteljük, hogy valamennyi körülötte lévő szomszédjának 1-es legyen az értéke (lásd 16. ábra). A művelet valamennyi pixelre történő végrehajtását követően az objektumok széle eltűnik, valamint azok a részek is, amelyek legfeljebb két pixel vastagságúak. Ennek következményeként fontos megjegyezni, hogy ez a művelet nem őrzi meg az objektumok topológiáját, mivel azok az erózió véges sokszori végrehajtása esetén előbb *szétesnek*, majd pedig teljesen eltűnnek.



(a) 4-szomszédság esetén (b) 8-szomszédság esetén

16. ábra. Erózió művelet

A kiterjedés művelet az erózió párjaként tekinthető, noha nem egymás inverzei. A művelet végrehajtásakor 4-, illetve 8-szomszédság esetén minden egyes 1 értékű pixelnek a szomszédság szerinti szomszédait 1 értékűvé módosítjuk (lásd 17. ábra). Ennek során az objektumok az éleik mentén megvastagodnak, aminek egyik következménye, hogy a művelet véges sokszori végrehajtása esetén az objektumokban lévő lyukak eltűnnek. Ebből az is következik, hogy ez a művelet szintén nem őrzi meg a topológiát.



(a) 4-szomszédság esetén (b) 8-szomszédság esetén

17. ábra. Kiterjedés művelet

Ahogyan korábban említésre került, a kiterjedés és erózió műveletek nem egymás inverzei, ami könnyen látható abból, hogy a kiterjedés által eltűntetett lyukakat az erózió nem képes helyreállítani, illetve fordított sorrend esetén, az erózió által törölt kisebb objektumok szintén nem állíthatóak helyre a kiterjedés művelettel.

4.8.2. Vékonyítás, drótváz

Ez a művelet egy speciális változata az eróziónak, mivel megőrzi a topológiát. Az eljárás lényege, hogy véges sokszori ismétlésével megkapjuk az objektumok drótvázát (csontvázát). Az eljárást 8-szomszédságot feltételezve alkalmazzák. Meg kell jegyezni, hogy ez az algoritmus bizonyos pontokban (több vonal keresztezésénél, illetve törtvonalak töréseinél) meghagyhat pontokat. Ezek eltávolításáról a következő szűrőnél lesz szó.

P_3	P_2	P_9
P_4	P_1	P_8
P_5	P_6	P_7

18. ábra. P_1 pont és szomszédainak címkézése.

Az algoritmus minden egyes vékonyításnál végigfut a kép összes pixelén, és minden egyes pixelt megvizsgál a szomszédjainak elemzésével. Az alábbi ábrán látható (lásd 18. ábra), hogy a P_1 vizsgált ponthoz képest órajárással ellentétes irányban járja be az algoritmus a szomszédos pontokat az elemzés során. A pontok bejárásakor a következőket számítja ki:

≤ $Z0(P_1)$: a nullából egyesbe való átmenetek száma a $\}P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_2|$ sorozatban.

- $Z0(P_1)$-et a P_1 pixel körbejárásával kapjuk.
- Ezt a számlálást gyakran arra is szokták használni, hogy egy vékonyított képen eldöntsék egy adott pontról, hogy vonal vége-e (ebben az esetben $Z0(P_1) = 1$).

≤ $NZ(P_1)$: P_1 nem nulla értékű szomszédjainak száma.

A fenti tulajdonságok alapján a vékonyítás algoritmusa a következőképpen néz ki:

1. Allokáljunk két képet, legyenek ezek $u_1(r, c)$ és $u_2(r, c)$ úgy, hogy méretük megegyezzen a kapott képünkével, azaz $u_0(r, c)$-vel. Legyen $u_2 = u_1 = u_0$.
2. Menjünk végig u_1 képen és töröljük a megfelelő pontokat u_2-ben az alábbiak szerint:
3. Minden egyes P_1 pontra számítsuk ki $Z0(P_1), NZ(P_1), Z0(P_2)$ és $Z0(P_4)$ értékeket.
4. Töröljük P_1-et, ha az alábbi feltételek **mindegyike teljesül**:

$$\begin{cases} 2 \geq NZ(P_1) \geq 6 \\ Z0(P_1) = 1 \\ P_2 \times P_4 \times P_8 = 0 \text{ vagy } Z0(P_2) \neq 1 \\ P_2 \times P_4 \times P_6 = 0 \text{ vagy } Z0(P_4) \neq 1 \end{cases}$$

5. Ha kész vagyunk u_1 kép feldolgozásával, álljunk meg, ha nem sikerült pontot törölnünk, egyébként pedig másoljuk u_2 tartalmát u_1-re és ugorjunk a 2. lépésre.

Az algoritmus kapcsán gyakran szokták emlegetni az úgynevezett középvonal transzformációt is (röviden MAT). Ez az objektum pontjaiból azt a vázat generálja, amelynek pontjai az objektum kettő vagy több legközelebbi határpontjától azonos távolságra helyezkednek el. A MAT hátránya egyrészt, hogy számításigényes, másrészt pedig a kontúrok kis hibái is erős torzulásokat okoznak. A MAT-al összehasonlítva a vékonyítással kapott drótvázról az alábbi tulajdonságok mondhatóak el:

≤ A drótváz egy **összefüggő** ponthalmaz.

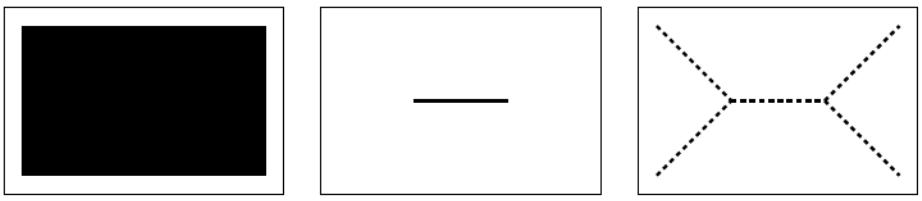
≤ A középvonalhoz hasonlóan a drótváz is tartalmaz minden elágazást.

- A drótváz követi az objektumok elnyúlt részeinek alakját.
- Más esetekben, az objektum és a drótváz közötti kapcsolat nem ilyen nyilvánvaló.

≤ A drótváz **általában hasonló** a középvonalhoz.

- Gyakran nincs különbség az objektumhoz tartozó drótváz és középvonal között. A két megközelítést tekintve alternatív módjai az objektumok drótváz reprezentációjának, ezért a középvonalat szintén szokás *drótváz*nak hívni.

≤ A drótváz **jelentősen eltérhet** a középvonaltól (például egy téglalap esetén is, lásd 19. ábra).



19. ábra. Téglalap alakú objektum transzformációja vékonyítással, illetve középvonal-transzformációval (MAT).

4.8.3. Morfológiai vékonyítás

A morfológiai vékonyítás szükségessége már az előző szűrő ismertetésekor előkerült. Ennek oka, hogy a vékonyító szűrő a topológia megőrzése miatt nem feltétlenül képes mindig olyan eredményt előállítani, ahol az objektum drótváza csupán egy pixel vastagságú. Erre jó példa a $2 * 2$ méretű négyzet, amelyet már nem tud tovább vékonyítani, illetve a törtvonalak töréseinél lévő Z alakú részek meghagyása. Ezek a későbbiekben

gondot okoznak a vonalak vektorizálásánál, ahol elvárt tulajdonság, hogy az elágazásoktól eltekintve minden pontnak legfeljebb két szomszédja legyen.

A probléma megoldására morfológiai vékonyítás alkalmazható. Ez a vékonyító módszer természetesen önmagában is képes létrehozni egy objektum drótvázát, azonban eredménye jelentősen eltér az előzőekben ismertetett vékonyító eljárásétól, és leginkább a MAT eredményéhez áll közel. Éppen ezért a morfológiai szűrőt a feldolgozás során csak a már előállított drótvázon szoktuk alkalmazni (lásd 21. ábra).

0	0	0
	1	
1	1	1

	0	0
1	1	0
	1	

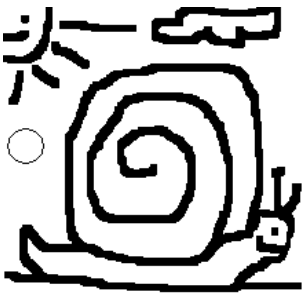
20. ábra. A morfológiai vékonyítás struktúráló eleme (bal oldalon), és 45•-kal elforgatott változata (jobb oldalon).

A morfológiai szűrők alapötlete egy struktúráló elem végigvitele a képen. Az adott képpont értékének esetleges megváltoztatásáról annak függvényében döntünk, hogy a pont megfelel-e a struktúráló elemnek. Ennek alapján különféle struktúráló elemek adhatóak meg, amelyek például az erózió, kiterjedés stb. eredményt adják. A számunkra fontos operátort Hit-Miss-nek hívják, amely bináris mintákat keres a képen. A Hit-Miss használatához vegyük a fentebbi képen látható struktúráló elemet (lásd 20. ábra). A morfológiai vékonyító megvalósításához vegyük az $\left\{L^0, L^1, \ldots, L^7\right\}$ elemeket, ahol L^i az L $i \circ 45$ fokkal elforgatott változata.

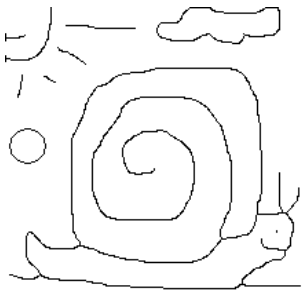
A vékonyítás a következő módon fogalmazható meg:

1. Másoljuk le az u_0 bemenő képet egy vele megegyező méretű u_1 képre.
2. $i \uparrow 0$
3. Vigyük végig L^i-t u_0 kép minden pontján úgy, hogy L^i középső eleme essen az aktuális képpont fölé mindig. Vizsgáljuk meg, hogy ilyenkor a struktúráló elemben lévő definiált (0 vagy 1) értékek alatt a képen ugyanolyan értékek találhatóak-e. Ha teljes egy pontban a struktúráló elemmel az egyezés a nem definiált részektől eltekintve, akkor a pontot u_1-ben töröljük.
4. $i \uparrow i + 1$
5. $u_0 \uparrow u_1$
6. Ha $i < 8$, akkor ugorjunk a 3. lépésre.
7. Ha történt törlés, ugorjunk a 2. lépésre.

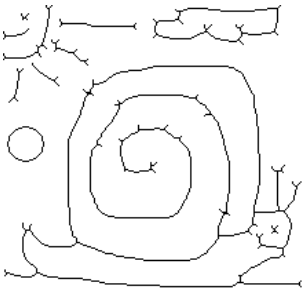
A fenti algoritmusból látható, hogy a vékonyítás egy lépése a kép 8 alkalommal való teljes végigjárását jelenti. Ebből következik az is, hogy a korábban ismertetett vékonyító eljárásnál jóval nagyobb a futási ideje.



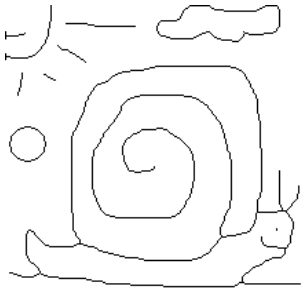
(a) Eredeti kép



(b) Vékonyítással kapott drótváz



(c) Morfológiai vékonyítással kapott drótváz



(d) Vékonyítás utáni morfológiai vékonyítás eredménye

21. ábra. Vékonyítás és morfológiai vékonyítás eredményei.

4.9. Színdetektáló eljárások

A térképek feldolgozásának egyik fő lépése a színdetektálás. Erre azért van szükség, mert szeretnénk az azonos színosztályba tartozó pixelekből álló egybefüggő területeket egyben kezelni, mint poligonokat stb. Ezt az eljárást előzik meg az előző fejezetekben tárgyalt képszűrők, hogy a képi hibákat eltávolítsuk, illetve csökkentsük, valamint a térképek színosztályai közötti távolságot növeljük. A magyar topografikus térképek esetében szerencsére viszonylag kis számú nyomtatott színről beszélhetünk, azonban találkozhatunk a térképekre jellemző sajátságos anomáliákkal. Ilyen problémát vet fel az egymás mellett lévő különböző osztályba tartozó pixelek közötti részen fellépő színátmenet is, vagy például a nyomtatási eljárásból adódó színkeveredés az egymásra nyomtatott objektumok esetében.

4.9.1. Egyszerű detektálás

A képpontok színdetektálásának egyik lehetséges módja, ha előre definiáljuk a térképen alkalmazott színeket (ezek lesznek a színosztályaink). Ekkor minden egyes képpontra kiszámítjuk, hogy a képpont színe melyik osztály színéhez van a legközelebb. RGB képek esetén ennek egyik lehetséges módja az euklideszi távolságok használata. Ekkor egy

P pontra a következő módon kapjuk meg színosztályának i indexét:

$$\underset{i \neg \}1..n|}{argmin} \sqrt{(R_i \quad P_R)^2 + (G_i \quad P_G)^2 + (B_i \quad P_B)^2}.$$

A módszer előnye, hogy könnyen megvalósítható és lineáris a futási ideje. Hátránya, hogy az RGB színmodellben a különböző komponenseket egyenrangúként kezeli, miközben az emberi szem látása ettől eltérő módon értelmezi a színeket. Másik anomáliához vezet például, ha egy pont színe két színosztály színeinek keveredéséből jön létre és egy harmadik osztály színéhez esik legközelebb. Ez azt jelenti, hogy ebben a megközelítésben figyelembe kellene venni a kevert színek esetében azon környező pontok színét, amelyeket teljes bizonyossággal tudtunk korábban meghatározni.

4.9.2. Réteg-alapú színdetektálás

Egy másik megközelítésként, az IRIS projektben már korábban sikeresen alkalmazott módszert kívánok bemutatni, amely a papír alapú térképek készítésének módját veszi figyelembe. A térképek nyomdai munkálatainál a nyomólemezeket két csoportba sorolhatjuk. Az első csoportba a felületeket tartalmazó lemezeket soroljuk, míg a másodikba az objektumokat tartalmazóakat. Egy csoporton belül annyi lemezt kell alkalmazni, ahány színre a színkeveréshez szükségünk van, illetve a további direkt színenként egyet-egyet. A nyomdai eljárásból adódóan az első csoporton belül a felületek, illetve a második csoporton belül az objektumok diszjunktak. A nyomdászat érdekessége, hogy a térképen az objektumok színhűségének növelésére nyomtatásukkor direkt színeket alkalmaznak (pl. szintvonalak esetén narancssárgás-barnát). Az elkészült lemezeket ezután úgy használják, hogy előbb a felületeket viszik a papírra és csak ezután az objektumokat.

Az imént vázolt nyomdai eljárás alapján az első csoportban létrejövő színkeverések szándékosak és a kívánt színeket képezik le. Ezzel szemben a második nyomtatási fázisnál az objektumok színe a festék papírra kerülésekor, a már papíron lévő festékek színével keveredik annak mértékében, hogy mekkora a festék fedő képessége. Ebből kiindulva elmondható, hogy a papíron létrejövő színek minden egyes pontban leírhatók egy felületi és egy objektum szín lineáris kombinációjaként.

A réteg-alapú színdetektáláshoz határozzuk meg a felületi színeket, legyenek ezek $f_1, \ldots, f_n$, valamint az objektum színeket $o_1, \ldots, o_m$. Ezután meg kell határoznunk az összes felület-objektum szín-párra, hogy melyek lehetséges lineáris kombinációja közelíti meg legjobban a térképen lévő pont színét. Ezt megoldhatjuk úgy, hogy lineáris kombinációnként egy minimalizálást végzünk. Legyen q_{ij} az i-edik felületi szín és a j-edik objektum szín távolságának négyzete, azaz egy három csatornás (RGB) kép esetében:

$$q_{ij} = \sum_{k=1}^{3} (f_{ik} \quad o_{jk})^2.$$

Alkalmazzuk a következő algoritmust p pixel színének meghatározására:

1. $mval \uparrow \quad +\in$

2. **for** $i \uparrow \ 1 \ldots n$ **do**
3. **for** $j \uparrow \ 1 \ldots m$ **do**
4. $(c, l, q) \uparrow \ (0, 0, q_{ij})$
5. **for** $k \uparrow \ 1 \ldots 3$ **do**
6. $c = c + (f_{ik} \quad p_k)^2$
7. $l = l + (f_{ik} \quad p_k) \circ (o_{jk} \quad f_{ik})$
8. **if** $q = 0$ **then**
9. $(mval, f_{index}, o_{index}, rm) \uparrow \ (0, i, j, 0)$
10. **continue**
11. $r \uparrow \quad l/q$
12. **if** $r < t$ **then** $r \uparrow \ 0$
13. **if** $r > 1$ **then** $r \uparrow \ 1$
14. $val \uparrow \ (q \circ r + 2 \circ l) \circ r + c$
15. **if** $mval > val$ **then** $(mval, f_{index}, o_{index}, rm) \uparrow \ (val, i, j, r)$
16. **if** $rm \sim t$ **then** $index \uparrow \ o_{index}$
17. **if** $rm < t$ **then** $index \uparrow \ f_{index}$

4.10. Színosztályozó eljárások

A projekt kezdeti szakaszában a térképek vektorizálásával kapcsolatos problémák vizsgálata és a használandó eljárások kiválasztása volt a cél. Noha a térképek feldolgozásának végső változatában nem kerültek alkalmazásra színosztályozó módszerek, mégis érdemes rájuk kitérni. A színosztályozó eljárások a színdetektáló eljárásokkal szemben nem igényelnek konkrét színeket, amelyekhez megpróbáljuk besorolni a térkép egyes pontjait. Itt a színosztályokat maga az eljárás határozza meg futása során. Általánosságban elmondható, hogy a színosztályozó módszerek aszimptotikusan jóval nagyobb futási idejűek, mint a színdetektáló eljárások.

4.10.1. Isodata klaszterezés

Az Isodata klaszterezésnek alapvetően két paramétere van a működése szempontjából: a tér dimenziója (d), valamint az, hogy hány osztályt szeretnénk elszeparálni (k). Esetünkben színes képeket tekintve a tér 3 dimenziós, mivel három színkomponensünk van. k értékét általában empirikus úton szoktuk megválasztani, térképek esetében azonban meg van az a könnyebbségünk, hogy tudjuk hány színt használtak a készítésekor. Amennyiben k értékét túl kicsinek választjuk, úgy bizonyos osztályok egyesítésre kerülnek, míg ha k értéke túl nagy, úgy a későbbiekben nekünk kell megtenni a megfelelő osztályok egyesítését. Bizonyos esetekben hasznos lehet, ha k értékét kicsivel nagyobbra állítjuk, mint a valós osztályok számát, mivel így a térben nő a szeparáció és a helyes besorolás esélye.

A következő eljárással optimális esetben legfeljebb M iterációt végezve az osztályok középpontját ϵ pontossággal kaphatjuk meg egy N pixelből álló kép esetén:

1. $iter \uparrow 0$
2. **for** $i = 1 \ldots k$ **do**
3. $\quad cnt_i \uparrow 0$
4. $\quad$ **for** $j = 1 \ldots d$ **do** $C_{i,j} \uparrow 0$
5. **for** $i = 1 \ldots N$ **do**
6. $\quad dist_{global} \uparrow +\infty$
7. $\quad$ **for** $j = 1 \ldots k$ **do**
8. $\quad\quad dist_{class} \uparrow \sum_{l=1}^{d} (P_{i,l} - c_{j,l})^2$
9. $\quad\quad$ **if** $dist_{class} < dist_{global}$ **then** $(dist_{global}, ind) \uparrow (dist_{class}, j)$
10. $\quad$ **for** $j = 1 \ldots d$ **do** $C_{ind,j} \uparrow C_{ind,j} + P_{i,j}$
11. $\quad cnt_{ind} \uparrow cnt_{ind} + 1$
12. $end \uparrow \Rightarrow$
13. **for** $i = 1 \ldots k$ **do**
14. $\quad$ **for** $j = 1 \ldots d$ **do** $C_{i,j} \uparrow C_{i,j}/cnt_i$
15. $\quad end \uparrow end \vee (\sum_{l=1}^{d} (C_{i,l} - c_{i,l})^2 < \epsilon)$
16. $\quad c_i \uparrow C_i$

17. $iter \uparrow \ iter + 1$

18. **if** $iter < M \vee !end$ **then goto 2.**

A fenti algoritmusnál RGB képek esetében d értéke 3, míg c_i egy szín, ami az i-edik osztály középpontját reprezentálja. Az egyes iterációkban a pontokat a könnyebbség kedvéért nem tesszük halmazba, amelynek az átlagát később ki kellene számolnunk. Az i-edik osztályba sorolt pixelek értékét C_i-ben összegezzük, számosságukat pedig cnt_i-ben tartjuk nyilván. Az algoritmus a kép pontjait az 5-11. lépésekben sorolja osztályokba, ahol az euklideszi távolságon alapuló besorolás a 6-9. lépésekben történik. A osztályok középpontjának *elmozdulását* a 12-15. lépésekben vizsgáljuk, ahol megköveteljük, hogy az eljárás megállásához egyik osztályközéppont se mozduljon el ϵ-nál nagyobb mértékben.

4.10.2. NeuQuant osztályozó

Ez a színosztályozó egy önszerveződő Kohonen neurális hálón alapul, amelyet Dekker [5] mutatott be. A módszer leginkább naturalisztikus képeken működik jól, célja pedig, hogy a képet egy indexes képpel helyettesítse, ahol az egyes képpontokhoz egy 0 és n 1 közötti értéket rendelünk. A gyakorlatban n értéke általában 256, így egy 256 színű palettát kell létrehoznunk a forráskép elemzése alapján.

Az eljárás a paletta létrehozásához egy n neuronból álló egy dimenziós, önszerveződő Kohonen neurális hálót használ. A betanítás során egyesével vesszük a kép minden m-edik pontját, és megvizsgáljuk melyik neuron van hozzá a legközelebb. Ezt a neuront, valamint a szomszédjait az aktuális pont értékéhez közelebb visszük, mégpedig a következő módon. Valahányszor egy neuron kiválasztásra kerül, megnöveljük a gyakoriságszámlálóját. Ennek értéke minél nagyobb, a neuron annál nehézkesebben mozog, vagyis kevésbé fog elmozdulni a következő alkalommal egy pont irányába.

Megvalósítás szempontjából a neuronok a következő adatokkal rendelkeznek: hely (szín), gyakoriság, erősítés (*bias*). Az erősítés itt azt jelenti, hogy a törtszámok kifejezésére az egész számok alkalmazása miatt a szín csatorna-intenzitásait felszorozzuk futás közben, amit a tanítás végeztével helyre kell állítanunk. A tanítás végeztével a kép feldolgozása már egyszerű feladat, hiszen minden képpontra megnézzük melyik neuron van hozzá legközelebb és annak indexét rendeljük a ponthoz értékként. A kép palettáját a neuronok hely (szín) értékei adják.

Amint említésre került, a módszer naturalisztikus képek esetén színkvantálásra alkalmazható jól. A lassú futási idő mellett másik hátránya, hogy legalább 64 neuront feltételez a helyes működéshez a cikk, miközben a térképeken legfeljebb 10 szín található. Ez könnyen áthidalható, ha addig vonjuk össze a két legközelebbi neuront az egyes összevonások alkalmával, amíg a kívánt méretűre nem csökken a paletta mérete.

5. nCoreAPI

A GeoFilterBank keretrendszer ismertetése során már említésre kerültek azok a főbb szempontok, amelyek segítségével a térképek feldolgozása hatékonyabbá tehető, illetve amelyek megkönnyítik a digitális képszűrő eljárások implementálását, rendszerbe illesztését. Ezen szempontok megvalósítására szolgál együttesen az nCoreAPI könyvtár. A következő három alfejezetben ezen könyvtár által nyújtott szolgáltatásokat mutatom be.

5.1. Bővítménykezelés és modulok közötti kommunikáció

A térképészetben a különböző időszakokban alkalmazott ábrázolási módszerek szerteágazóak, ezért ezeknek a térképeknek a feldolgozása, ha kicsiben is, de eltér. Emiatt az eltérés miatt gyakran alkalmasabb egy, a célnak megfelelő újabb szűrő implementálása. Ebből a tényből kiindulva célszerűnek látszott egy moduláris rendszer elkészítése. Egyszerűségi okokból a modulokban megvalósított funkciókat távoli eljárás hívással (RMI) a legkönnyebb elérni. Ilyenkor ugyanis nincs szó modulokon belüli objektumokról, amelyeket példányosítani kellene, illetve az élettartamukat felügyelni. Ennek következménye, hogy az állapotleírásokat a keretrendszernek kell kezelnie, amihez szükséges egy olyan kommunikációs lehetőség, amely bármilyen formátumú adatot képes forgalmazni a keretrendszer és az adott modul között. Ennek megvalósítására egy olyan szerializáció tűnt a legalkalmasabbnak, amely faszerkezetben tárolt adatokat képes bájtsorozattá konvertálni, illetve azt vissza. Az adatok fában történő reprezentációjához az XML felhasználása bizonyult előnyösnek, mivel a fa csúcsai címkézettek, így egyrészről rugalmasan helyezhető el benne tetszőleges adat, másrészt alkalmas az adatok hierarchikus tárolására. Az XML faszerkezet másik fontos tulajdonsága, hogy tartalma szelektíven feldolgozható, így a modulok által elhelyezett állapotleírókat a keretrendszer figyelmen kívül hagyhatja, miközben egy későbbi eljáráshívás alkalmával felhasználhatja ezeket ismételten (pl. workflow-k kötegelt feldolgozásánál).

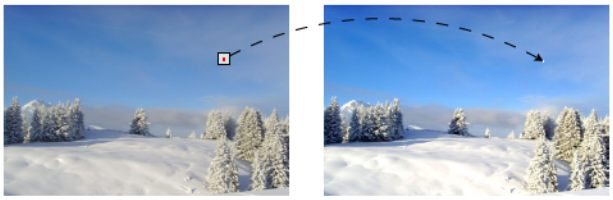
5.2. Szűrőalgoritmusok automatikus párhuzamosítása

A GeoFilterBank térképinterpretációs rendszer implementálásának idejében, illetve a hozzá kapcsolódó kutatások során is már léteztek többprocesszoros gépek, manapság pedig szinte csak ilyeneket lehet vásárolni. Mindezek ellenére csekély törekvés látszik az iparban arra nézve, hogy a térinformatikához kapcsolódó képszűrő algoritmusokat a párhuzamos környezetbe átültessék. Ennek egyik lehetséges oka abban keresendő, hogy párhuzamos algoritmusokat készíteni jóval bonyolultabb, és nagyobb tudást igényel. Látható, hogy a korábbi fejezetben nagy számú képfeldolgozó eljárás szerepel, ezek egyesével történő párhuzamosítása pedig nagy kihívás lett volna. Ezért úgy gondoltam, hogy bizonyos szűrőtípusokat célszerű API szinten párhuzamosítani. Ez azzal az előnnyel jár, hogy a szűrőket továbbra is elegendő szekvenciális algoritmusokként implementálni.

Párhuzamos környezetben az algoritmusok párhuzamosításának legkönnyebb módja, ha a feladatot képesek vagyunk diszjunkt részfeladatokra felbontani. Mivel sok szűrő a

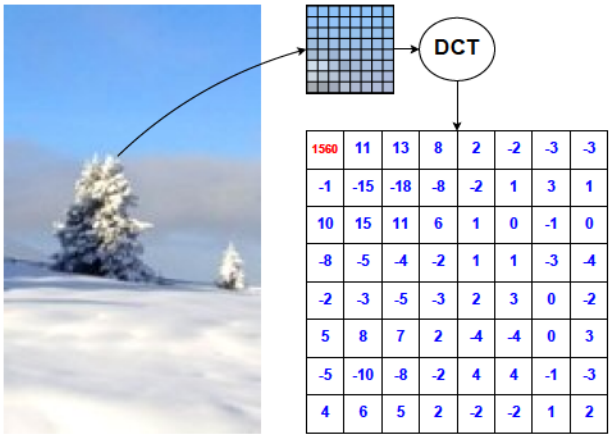
feldolgozás során a szomszédos pixelek értékét is felhasználja, így a diszjunkt részfeladatok meghatározása bonyolultabbá válik. Ezek alapján az algoritmusok párhuzamosításának módját leginkább a memóriahasználat módja szabja meg. Ebből kiindulva négy szűrőcsoportot különböztettem meg: lineáris, blokk, rekurzív és véletlen memóriaelérésű szűrők.

A lineáris memóriaelérésű szűrők a kimeneti kép pixeleinek sorfolytonosan vagy oszlopfolytonosan adnak értéket. A keletkező pixelértéket, a vele azonos pozícióban lévő forráspixel és annak környezete alapján számítjuk ki (lásd 22. ábra). Ilyen szűrők például a Fényerő, Kontraszt, illetve a doboz szűrők, mint például a Medián, Laplace, stb.



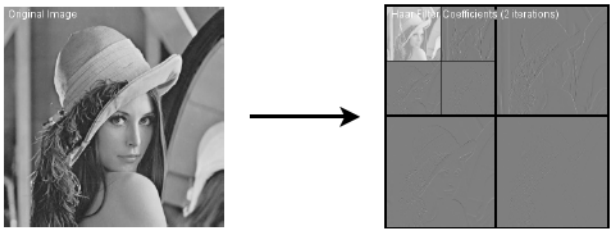
22. ábra. Kontraszt szűrő

A blokk típusú szűrők a képet blokkonként dolgozzák fel, ahol a blokkok egymástól fizikailag különálló téglalap alakú képrészletek. Egy blokk kiszámításához csak a neki megfelelő blokkban lévő pixelek értékét használjuk fel. Tipikus blokkszűrő például a DCT (diszkrét koszinusz transzformáció, lásd 23. ábra), amely a képet 8x8-as blokkokra osztja.



23. ábra. Diszkrét koszinusz transzformáció (DCT)

A harmadik csoportba azokat a szűrőket soroltuk, amelyek a képet több lépésben dolgozzák fel. Egy n lépésből álló feldolgozás i. lépésében az i 1. lépés képi eredményét $2^i * 2^i$ (vagy $2^{n\ i} * 2^{n\ i}$) méretű blokkokra osztjuk, és ezeket egymástól függetlenül dolgozzuk fel. Főként a wavelet szűrők működésére jellemző ez a rekurzív megközelítés (lásd 24. ábra).



24. ábra. Haar Wavelet szűrő

A véletlen-memóriaelérésű szűrők a forráskép pixeleit véletlen sorrendben választják ki – sok esetben statisztikai céllal. Ezek a szűrők gyakran több menetesek, és csak a második, vagy későbbi menetben adnak értéket az eredménykép pixeleinek (pl. neurális hálóra épülő színosztályozás).

5.2.1. A kép logikai particionálása

Feltesszük, hogy a párhuzamosítandó szűrők paraméterként kaphatnak egy téglalap alakú területet, ahol a képen végre kell őket hajtani. A képet érdemes általában annyi részre osztani, ahány processzorral rendelkezik a számítógép. Ezt követően a képrészleteket párhuzamosan fogjuk szűrni. Az előző fejezetben említett véletlen memóriaelérésű szűrőkkel nem foglalkozunk, mivel ezeknél nem határozhatóak meg átlapolásmentes képrészletek. A képparticionálás és a szűrőpárhuzamosítás a következőkben memóriaelérési fajtánként kerülnek ismertetésre.

5.2.2. Kép-particionálás lineáris-memóriaelérésű szűrők esetén

Feltesszük, hogy a lineáris szűrő sorfolytonosan (vagy oszlopfolytonosan) dolgozza fel a képet, valamint megengedett, hogy a pixelértékek kiszámításához felhasználásra kerüljön annak r sugarú környezete is (például egy $3 * 3$-as kernelű szűrő esetén $r = 1$). A pixel környezetét is felhasználó szűrők működése hibásnak tekinthető, ha párhuzamosan futó folyamataik ugyanazt a memóriaterületet használják éppen (írási/olvasási konkurencia). A diszjunkt párhuzamos programok helyességének szükséges feltétele, hogy a folyamatok interferencia mentesek legyenek, ezért a párhuzamosítás alapötlete az, hogy a képet diszjunkt részekre osztjuk (partícionáljuk) és ezeket a képrészleteket párhuzamosan dolgozzuk fel.

Mivel egy ablakos szűrőfolyamat "hatásköre" túlnyúlik az általa feldolgozott képrészleten, ezért a képrészlet határától kifelé, r távolságon belül olvasási konkurencia lép fel a szomszédos képrészleteken dolgozó szűrőfolyamatokkal. Ennek elkerülésére először a képrészletek közötti r vastagságú *csíkok* feldolgozását végezzük el (a vízszintesek és a függőlegesek feldolgozása külön-külön párhuzamosítható), és majd csak ezután dolgozzuk fel párhuzamosan a fennmaradó képrészleteket.

A gyakorlatban hatékonysági okokból a párhuzamosan feldolgozandó képrészletek száma a számítógép processzorainak N számával egyezik meg. Az alábbi FELOSZT-CPU

algoritmussal a képet $N = m \circ n$ részre osztjuk úgy, hogy a képrészletek a lehetőségekhez mérten azonos területű téglalapok legyenek, ahol w a kép szélessége és h a magassága:

FELOSZT-CPU(w, h, N, m, n)

1. $m \uparrow \quad \overline{N}$
2. **while not** $m \backslash N$ **do** $m \uparrow \quad m \quad 1$
3. $n \uparrow \quad N/m$
4. **if** $w < h$ **then csere**(m, n)

A kapott m, n értékeket felhasználva a kép szélességét n-nel, magasságát pedig m-mel osztva megkapjuk az "ideális" felosztást. A képrészletek és csíkok pozícióit a következő módon kapjuk:

FELOSZT($w, h, m, n, r, BlokkX, BlokkY, CsíkX, CsíkY$)

1. $BlokkX[0] \uparrow \quad 0$
2. $BlokkY[0] \uparrow \quad 0$
3. **for** $j = 1$ **to** $n \quad 1$ **do**
4. $\quad CsíkX[j \quad 1] \uparrow \quad j \circ (w \quad 2 \circ r)/n \quad (r/2)$
5. $\quad BlokkX[j] \uparrow \quad CsíkX[j \quad 1] + r$
6. **for** $i = 1$ **to** $m \quad 1$ **do**
7. $\quad CsíkY[i \quad 1] \uparrow \quad i \circ (h \quad 2 \circ r)/m \quad (r/2)$
8. $\quad BlokkY[i] \uparrow \quad CsíkY[i \quad 1] + r$

A létrehozott csíkok vastagsága r, az $n \quad 1$ függőleges csík közül a j-edik x koordinátája $CsíkX[j \quad 1]$, az $m \quad 1$ vízszintes csík közül pedig az i-edik y koordiánátája $CsíkY[i \quad 1]$. Az (i, j) cimkéjű processzorral feldolgoztatandó képrészlet az eredeti kép alábbi része:

$$[BlokkX[j \quad 1], CsíkX[j \quad 1] \quad 1] * [BlokkY[i \quad 1], CsíkY[i \quad 1] \quad 1],$$

(amely a képrészlet pontjainak $\mathbb{N}^2$-beli koordinátáit tartalmazza, mint vízszintes és függőleges intervallumokból képzett direktszorzat).

A szűrő párhuzamosításának vizsgálatakor látható, hogy az N processzor mindegyike a kép N-ed részével foglalkozik, így a párhuzamosított algoritmus által végzett munka megegyezik a szekvenciális algoritmus munkájával, vagyis a párhuzamos algoritmus munkaoptimális. Az eljárás segítségével tehát a végrehajtási időt az N-ed részére sikerült csökkenteni (eltekintve a csíkok kiszámításához tartozó minimális többletidőtől).

5.2.3. Kép-partícionálás blokk-memóriaelérésű szűrők esetén

A blokk alapú szűrők a képet logikailag $u * v$ méretű részekre osztják, melyeket egymástól függetlenül dolgoznak fel. Párhuzamosítás szempontjából ez azt jelenti, hogy az egyes blokkok feldolgozása - a memóriahasználatot tekintve - diszjunkt. A lineáris szűrőkkel ellentétben a blokkon belüli írási és olvasási műveletek jól meghatározott sorrendben kerülnek végrehajtásra. Az esetleges konkurrens memóriaelérést a blokk típusú szűrők saját maguk kezelik, ezért ezzel a nehézséggel nem kell foglalkoznunk. A blokkfeldolgozó algoritmusokat az előző fejezetben bemutatott FELOSZT függvénnyel tehetjük alkalmassá arra, hogy párhuzamosítsuk azokat. A kép egyes részeinek processzorokhoz rendelése során a blokkok diszjunktsága miatt $r = 0$ lesz. A kép particionálásakor ügyelnünk kell arra, hogy a részek határai blokkhatárra essenek. Ezt úgy tehetjük meg legegyszerűbben, ha a FELOSZT függvényt a következő módon hívjuk meg:

$$\text{FELOSZT}(w/u, h/v, m, n, r, BlokkX, BlokkY, CsíkX, CsíkY)$$

A FELOSZT függvény eredményének segítségével a következő módon definiálható az (i, j) cimkéjű processzor által feldolgozandó képrészlet:

$$[BlokkX[j \quad 1] \circ u, CsíkX[j \quad 1] \circ u \quad 1] * [BlokkY[i \quad 1] \circ v, CsíkY[i \quad 1] \circ v \quad 1]$$

Mivel a lineáris szűrőknél alkalmazott technika került ismét felhasználásra, ezért szintén továbbra is fennállnak a munkaoptimalitásra és a futási időre vonatkozó korábbi megállapítások.

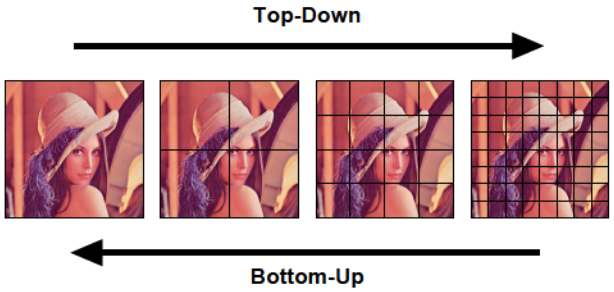
5.2.4. Kép-particionálás rekurzív szűrők esetén

Mielőtt konkrét algoritmust mutatnék be, tegyük fel, hogy a kép megfelel a végrehajtandó rekurzív szűrő feltételeinek (pl. a kép méretei 2 hatványok). A rekurzió szempontjából létezik *top-down*, illetve *bottom-up* megközelítés is (lásd 25. ábra). Előbbi esetben a szűrő minden egyes lépésben a kép blokkjait $k * l$ darab blokkra ossza, míg utóbbi esetben minden szomszédos $k * l$-es blokkcsoportot egy-egy blokkba szervez. *Bottom-up* megközelítés esetén az $0 \geq i.$ lépésben a blokkok mérete $k^i * l^i$, míg *top-down* esetben $\frac{w}{k^i} * \frac{h}{l^i}$.

A két esetet együttvéve a következő függvényt definiálhatjuk:

REKURZÍVSZŰRŐ$(src, dest, w, h, k, l, I, I_{kezd}, szűrő)$

1. $sign \uparrow \quad sgn(I)$
2. $s \uparrow \quad (sign + 2)/2$
3. **for** $j = \backslash_{kezd} \backslash$ **to** $\backslash \backslash$ **do**
4. $\quad w^{\infty} \uparrow \quad w^s \circ k^{1 \quad s \quad j \circ sign}$



25. ábra. *Top-down* és *Bottom-Up* megközelítés a rekurzív képszűrőknél, ahol $k = l = 2$

5. $h^{\infty\uparrow}$ $h^s \circ l^1$ *s j∘sign*

6. **for** $P_{A,B}$ **in parallel** $[1, 1]$ **to** $[w/w^{\infty}, h/h^{\infty}]$ **do**

7. $p \uparrow$ $(A \circ w^{\infty}, B \circ h^{\infty})$

8. *szűrő*$(src + p, dest + p, w^{\infty}, h^{\infty})$

A fent megadott függvény *src* és *dest* paraméterei az eredeti képet, illetve a létrehozandó képet jelentik. *szűrő* egy paraméterként kapott függvény, amely egy szekvenciális szűrőt tartalmaz és amely első két paraméterben kapja meg a forrás- és eredménykép címét, valamint az utolsó két paraméterében kapja meg a feldolgozandó képrészlet méretét. Az I_{kezd} paraméter bevezetésének az az eset az oka, amikor egy *top-down* rekurzív szűrő már az első lépésben a képet blokkokra osztva szeretné használni. Szintén szükséges az I_{begin} használata, ha a *bottom-up* típusú szűrőket egy minimális blokkmérettel akarjuk indítani. Látható, hogy ha $k = l = 1$, akkor ez a szűrő egymás utáni \∀\ \∀$_{kezd}$\-szer történő végrehajtását jelenti. Amennyiben $I = 0$, akkor az eredeti, szekvenciális szűrőt kapjuk. Ha a függvényt a $k = u, l = v$, valamint $I = I_{kezd} = 0$ értékekkel hívjuk meg, (ahol u és v a blokkszűrőknél használt blokkok mérete), akkor egy párhuzamosított blokkszűrőt kapunk (amely azonban nem hatékony, mivel minden blokkhoz egy új folyamatot indít el).

A párhuzamosítási módszer hatékonyságát vizsgálva, ha N processzort használunk, akkor például *top-down* esetben az összes processzor csak az i. menettől kezdve lesz kihasználva, amikor $N \geq k^i \circ l^i$. A gyakorlatban megfigyelhető, hogy a processzorok teljes kihasználtsága esetén ront a hatásfokon az, ha túl nagy számú folyamat fut egy időben. Ez az eset akkor áll fenn, ha N nagyságrendileg kisebb, mint a blokkok száma, azaz $N \leftarrow k^i \circ l^i$.

5.2.5. Egy általánosított párhuzamosítási módszer

A korábban bemutatott párhuzamosítási módszereket felhasználva a következő függvény egy általános párhuzamosítási eljárást [23] ad meg:

PÁRHUZAMOSSZŰRŐ$(src, dest, w, h, u, v, k, l, I, I_{kezd}, szűrő)$

1. **if** $I \neq 0$ **then**
2. RekurzívSzűrő($src, dest, w, h, k, l, I, I_{kezd}$, *szűrő*)
3. **return**
4. Feloszt($w, h, m, n, r, B_x, B_y, S_x, S_y$)
5. **for** P_i **in parallel** 0 **to** m 2 **do** *szűrő*($src + (0, S_y[i]), dbuf + (0, i \circ r), w, r$)
6. **for** P_j **in parallel** 0 **to** n 2 **do** *szűrő*($src + (S_x[j], 0), dbuf + (j \circ r, 0), r, h$)
7. **for** $P_{i,j}$ **in parallel** $(0, 0)$ **to** $(m$ 2, n 2) **do**
8. $w^{\infty} = S_x[j + 1]$ $B_x[j]$ 1
9. $h^{\infty} = S_y[i + 1]$ $B_y[i]$ 1
10. $p = (B_x[j], B_y[i])$
11. *szűrő*($src + p, dest + p, w^{\infty}, h^{\infty}$)
12. **for** P_i **in parallel** 0 **to** m 2 **do** *másol*($dbuf + (0, i \circ r), src + (0, S_y[i]), w, r$)
13. **for** P_j **in parallel** 0 **to** n 2 **do** *másol*($dbuf + (j \circ r, 0), src + (S_x[j], 0), r, h$)

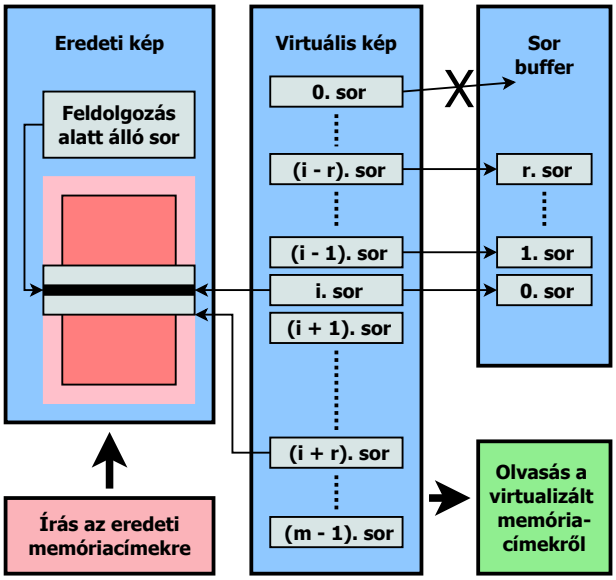
A *másol* művelet az első paraméterében kapott helyről a második paraméterben kapott helyre másolja azt a képrészletet, melynek méretét az utolsó két paraméter definiálja.

5.3. Virtuális memóriakezelés

A nagy felbontású és méretű képek esetén egyik célunk, hogy helyben legyen végrehajtható az adott digitális képszűrő. Ez csak azon szűrők esetében okoz gondot, amelyek az adott képpont értékének kiszámításához más képpontok értékét is felhasználják. Az előző alfejezetben ezen szűrők egy csoportját memóriaelérésük alapján lineáris szűrőknek neveztük. A továbbiakban ilyen lineáris szűrőkhöz ad segítséget az API-hoz kifejlesztett virtuális memória támogatás.

Ahhoz, hogy a forrás és a célkép memóriaterülete megegyezhessen, feltesszük, hogy a lineáris szűrő sorfolytonosan (vagy oszlopfolytonosan) dolgozza fel a képet. Emellett megengedjük, hogy a pixelek értékének kiszámításához felhasználjuk annak r sugarú környezetébe eső pontok értékét is (például egy $3 * 3$-as kernelű szűrő esetén $r = 1$). Megjegyzendő, hogy bizonyos szűrők ablaka speciális alakú lehet, ilyenkor azonban a pont és tőle a legtávolabb szükséges pont közötti távolság megfelelő r-ként.

Látható, hogy a pixel környezetét is felhasználó szűrők hibásan működnek, ha ugyanarra a memóriaterületre írnak, mint amelyről olvasnak. Ezt a problémát virtuális kép



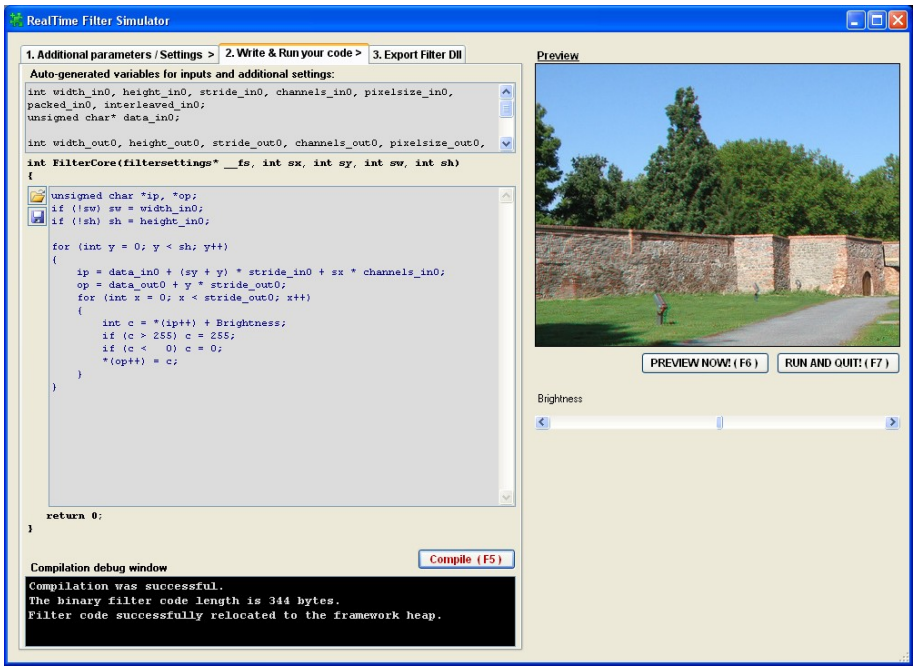
26. ábra. Virtuális kép használata az írás/olvasási konfliktus kezelésére a szűrő helyben történő végrehajtásához.

használatával kerülhetjük el. A virtuális kép elkészítéséhez az eredeti képet és egy ciklikus sorbuffert használunk (lásd 26. ábra). A sorbuffer $r + 1$ sort tartalmaz, ahol az egyes sorok hossza megegyezik az eredeti kép egy sorának bájtban mért méretével. A virtuális kép egy mutatótömböt tartalmaz, amely a kép sorait reprezentálja. Kezdetben a tömbnek az i. eleme az eredeti kép i. sorára mutat. Mikor a szűrő új sort kezd el feldolgozni, a virtuális képen léptetnie kell a következő sorra. Kezdetben az első léptetéssel bekerül a kép első sora a bufferbe, majd a második, stb. Az $r + 2$. és későbbi léptetéseknél a sorbufferben a legrégebben bekerült sort felülírjuk az eredeti kép aktuális sorával. Mindeközben minden egyes léptetésnél a virtuális kép mutatótömbjének aktuális sorhoz tartozó elemét beállítjuk a sorbufferben felülírt sor memóriacímére.

Az eljárás előnye, hogy segítségével egy szűrőt egyaránt használhatunk úgy, hogy az eredmény kép az eredeti képet írja felül, valamint hagyományos módon is, ha az írandó memóriacímet egy új kép címére állítjuk. Ez a megoldás tulajdonképpen a kép múltbéli állapotát szimulálja az olvasás számára legfeljebb addig, amíg arra szükség lehet, miközben a távoli jövőben felülírandó sorok bufferelésétől eltekintünk. A lineáris szűrők könnyebb implementálásához a módszer kiterjeszthető úgy, hogy a szűrőknek a kép szélét ne kelljen speciális esetként külön kezelniük. Ehhez a virtuális képet körben kiterjeszthetjük a kép szélén lévő pixelek duplikálásával. Ebben az esetben a szűrőnek az első sor feldolgozása előtt a virtuális kép sorait r-szer kéne léptetnie.

5.4. Valós idejű szűrő szimulátor

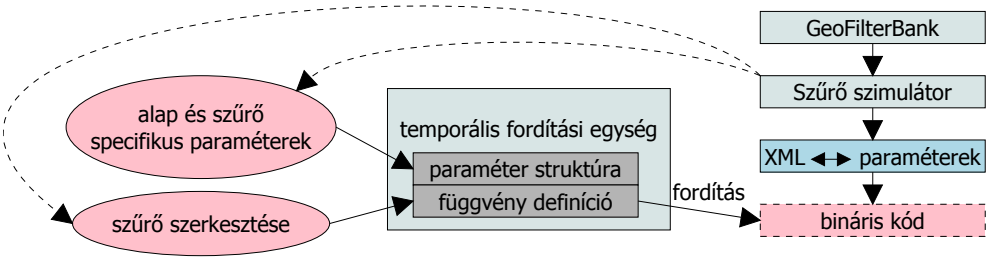
A GeoFilterBank rendszer tervezése és fejlesztése közben el kellett dönteni, hogy mely képszűrő algoritmusok képezzék a rendszer alapját. Ehhez előzetesen sok szűrőt kellett volna kipróbálni, hogy azokból a lényegeseket ki lehessen válogatni. Egy szűrő kipróbáláshoz azt implementálni kell, ami csak akkor hasznos, ha a szűrő a későbbiekben használatba is kerül. Tekintve, hogy a rendszer nagyjából 40 szűrőt tartalmaz alapkészletként, látható, hogy ez igen fáradtságos munka lett volna. Ennek elkerülésére egy teljesen új ötlet merült fel. Ahelyett, hogy külön-külön implementálásra kerültek volna a szűrők, egy teljesen általános szűrőburkot hoztam létre, amiben a szűrőmag tetszőlegesen cserélgethető akár fejlesztőkörnyezet nélkül is, amint azt a lentebbi kép mutatja (lásd 27. ábra).



27. ábra. A valós idejű szűrő szimulátor szerkesztő ablaka

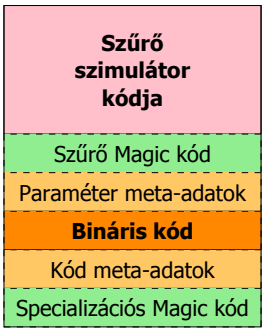
A szűrőburok biztosítja, hogy a rendszer és a szűrő között a paraméterek átadása megtörténjen, valamint kezeli a távoli metódushívásokat. A képet leíró közös paramétereken felül lehetséges további paraméterek bevezetése is, mivel a rendszer az XML-ben tárolt ezen részeket csak másolja, de nem értelmezi. A szűrőburokhoz a szűrőmagot C++ nyelven írhatjuk meg úgy, hogy a paramétereket egy rekordon keresztül érhetjük el. Az XML adatok és a rekord adatok között az információcsere automatikusan történik. A megoldás lényege, hogy a szűrőmag mindig azonos számú és típusú paraméterrel kerül meghívásra. Ez teszi lehetővé, hogy a szűrőmag, mint különálló függvény akár külön fordítási egységbe is kerüljön. A függvény a rekordtípussal együtt önállóan is lefordítható íly módon akár egy GNU C++ fordítóval. A kész objektumfájlból ezután a bináris kód kinyerhe-

tő, visszaolvasható a memóriába és az ismert paraméterezés segítségével meg is hívható (lásd 28. ábra). A problémát leegyszerűsítendő, a paraméterek tömbjére mutató pointer kerül csak átadásra és ezen tömböt inicializálja a burok. A függvény bináris kódjának visszanyerésére kódvisszafejtést alkalmaztam.



28. ábra. A szűrő szimulátor működésének blokk diagramja

A szimulátorban kipróbált szűrők forráskódjai C++ nyelvi forrásfájlba menthetőek, valamint arra is lehetőség van, hogy a kész szűrőt teljes egészében egy önálló modulként exportáljuk, majd azt később közvetlenül használjuk a keretrendszerben. Ehhez az MP3 fájloknál a szerzői és egyéb adatok tárolására használt módszer (ID3V1 szabvány) ötletét alkalmaztam, azaz ilyenkor a dll fájl végére másolódik a bináris kód a meta-adataival együtt. A fájl végére egy azonosító kerül, hogy a szűrő képes legyen futtatásakor felismerni önmagáról, hogy egy specializált változatáról van-e szó. A specializált fájl felépítése az alábbi képen látható (lásd 29. ábra).



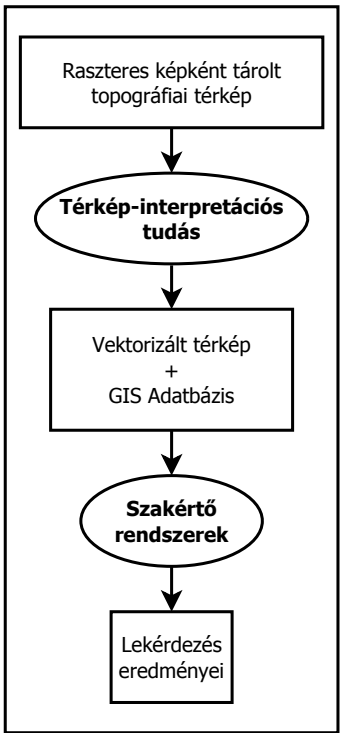
29. ábra. A szűrő szimulátor által specializált szűrőfájl felépítése

6. Magyar topográfiai térképek interpretációja

A térképekkel kapcsolatos szakértői tudásnak két szintjét különböztethetjük meg (lásd 30. ábra ellipszisben feltüntetett elemeit). Az első szinten foglal helyet a térképek interpretálásához szükséges szakismeret, amelynek segítségével a papír alapú szkennelt to-

pográfiai térképből elkészíthetjük annak vektoros megfelelőjét. A térképről kiolvasható információ alaprétege általában egy szabványos fájlban nyer elhelyezést, míg a finomabb összefüggéseket relációs adatbázisban szokás tárolni, a vektorizált állományhoz kapcsolódóan. A szakértelem másik szintjét azok az ismeretek alkotják, amelyekkel a vektoros adatmodell és a hozzá kapcsolódó adatbázis-tartalom alapján képesek vagyunk különböző következtetéseket levonni.

Az utóbbi témakörben már több kutatást végeztek [4, 1]. Tipikusan olyan kérdések tartoznak ide, mint pl. egy folyó áradásakor mely területeket borít majd nagy valószínűséggel a víz, vagy mennyire lehet gazdaságos egy feltárt olajmező kitermelése. A szakértő rendszerek a számadatokból álló válaszokon kívül gyakran alkalmaznak vizuális megjelenítést is a jobb érthetőség kedvéért [2].



30. ábra. Tudásábrázolás a geoinformatikában.

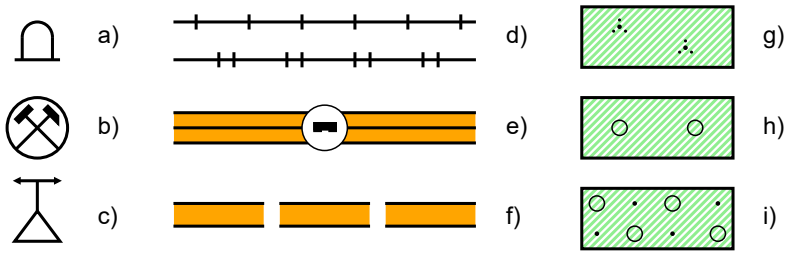
Fontos, hogy egy szakértő rendszer válaszainak jósága nem csak a szakértő rendszerben alkalmazott következtetési szabályoktól függ [3], hanem nagy részben attól is, hogy milyen minőségű tárolt adatokon hajtják őket végre [7, 18]. Ez is a térkép-interpretációs tudás fontosságát emeli ki. A mi esetünkben a tudásalapú megközelítés az előbb említett első szintre (a raszter-vektor konverzióra) vonatkozik.

A jó minőségű és automatizált vektorizációval kapcsolatban meg kell gondolni a következőket. A raszteres formájú térképen is találhatók számok, feliratok és más tényszerű adatok, de az információ jelentős részét egy olyan speciális képi forma tartalmazza, amelyet egy szakember ért igazán mélyen. A térkép vektorizált változata már nem tartalmaz

ilyen jellegű, interpretálásra váró összefüggéseket, hanem csak számszerű és karakteres adatokat. Az információvesztés szükséges velejárója a vektorizálásnak, ezért meg kell fontolni a konverzió mélységét. Ami a képi tartalmak automatikus interpretálását illeti, az erős képfeldolgozási eszközöket igényel, és az esetek nagy részében nem veheti fel a versenyt az emberrel szemben. Ezért az automatikus felismerés szintjét is körültekintően kell meghatározni. A következőkben a térképészetben alkalmazott raszter-vektor konverzió tudásalapú megközelítése kerül bemutatásra [26]. Ezen belül kerülnek felsorolásra a topográfiai térképeken alkalmazott jelkulcsi-elemek csoportjai, és az, hogy milyen algoritmussal lehet azokat felismerni. A fejezet végén sor kerül az interpretációban alkalmazott ismeretek tudásbázisba történő szervezésére is.

6.1. Térképi jelkulcsi elemek

A térképek nyomtatott változatainak értelmezése és számítógépre vitelének folyamata alapvetően a térképen szereplő jelkulcsi elemek értelmezésére és feldolgozására vezethető vissza. A feladat számítógépes megoldásához először is jól kell ismernünk a térképet magát, azon belül is a térképi jelkulcsot. Alapos tájékozódás céljából fordulhatunk a [13] könyvhöz. Noha az emberi kognitív folyamatok sohasem tisztázhatók teljesen, mégis valamelyest ismernünk kell azt, hogy a térképolvasó szakember hogyan értelmezi a rajzos információt. Az ember elsősorban pontszerű, vonalas és mintázott felületű jelkulcsi elemeket keres és különböztet meg a szemével (lásd 31. ábra). A térképet jól ismerő ember előtt a finomabb, rejtett információ is feltárul, például a keresztező többszintű útvonalak takarási viszonyai. Az emberi elme képes absztrahálni is, amikor például a textúrázott felület konkrét mintázatától elvonatkoztat és csak a felület alakját vizsgálja. Az emberi szem korrekcióra is képes, például különböző egymásra nyomott színrétegek árnyalatinak meghatározásakor.



31. ábra. Pontszerű jelkulcsok: a) emlékmű, szobor, b) működő akna bejárata, c) meteorológiai állomás. Vonalas jelkulcsok: d) közúti villamosvasutak, e) autópálya, (autósztráda), út menti segélykérő telefon, f) épülő műút. Felületi jelkulcsok: g) sűrű bozót, h) gyümölcsös, i) bokros gyümölcsös.

A négy különböző objektumtípus - a pont, vonal, felület és név - szinte tetszőleges példáján keresztül jól szemléltethető a térképolvasási folyamat, és a tudásalapú objektumfelismerés bonyolultsága. A térkép értelmezésének lépései megoldhatónak látszanak az

informatika eszköztárával, de a fejlett emberi látással párosult térképolvasási ismeretek eredményessége várhatóan nem érhető el a jelen kor informatikai technikáival.

A "pontszerű" elem leggyakrabban egy térképjel, egy kicsiny felületű grafika, egy alakzat, amely gyakran a tényleges fizikai méreténél nagyobb kiterjedésű felületet fed le, mégis megmondható, hogy valójában mi a vonatkoztatási pontja. Hasonlóan, a "vonalas" elemek is lefedhetnek a térképen a valós méretüknél szélesebb sávot. Például egy autópálya esetében egy "nulla vonalvastagságú" középvonal reprezentálhatja egy adatbázisban az útpálya eszmei helyét. Az adatbázis a térképi grafikai vonaltulajdonságokon túl természetesen tartalmazhatja adatként a valós fizikai paramétereket is: útszélesség, teherbírás, burkolat (beton, aszfalt) stb.

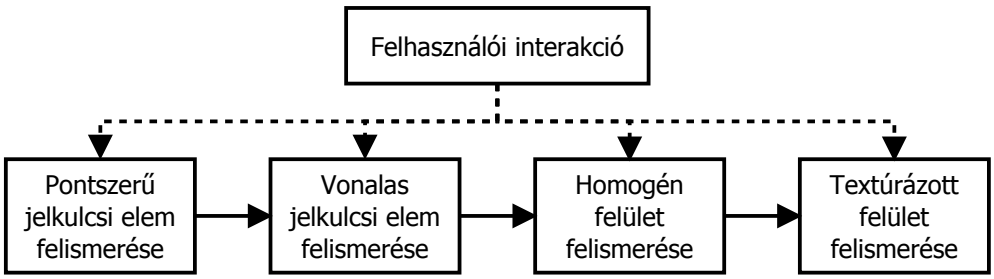
A takarás a térben különböző szinteken futó vonalas objektumok, tereptárgyak (tipikusan utak, vasutak, vezetékek) látványának képi összegzése, amely az egyes elemek folyamatosságának megszakadásával jár a térképi ábrázolásban. Informatikai szempontból azonban, a valóságnak megfelelően, feltételezzük az elemek folytonosságát. A felületi elemeknél már nem ennyire egyértelmű az értelmezés. Egy felület megszakadása a térképen általában nem jelenti a valós objektum megszakadását (pl. vízfelületként ábrázolt folyót átívelő híd esetén), de jelentheti azt is (pl. egy folyón létesített duzzasztógát esetében, ami akár egyben a vízfelületek szintbeli elkülönítését is eredményezheti).

A névrajzi elemek felismerése és értelmezése két szinten történik. Alapvetően, fel kell ismerni az összefüggő karaktersorozat által kifejezett elnevezést (ehhez szükség lehet a háttérben egy névadatbázisra). Az értelmezés a név és az általa jelölt tereptárgy közötti megfelelő kapcsolat megteremtését jelenti. A névrajzi elemek felismerése egy rendkívül bonyolult folyamat, mivel a betűk mérete, típusa, állása, távolsága sokféle lehet (például egy folyó sodorvonalát követi, vagy egy hegy vonulatát). A topográfiai térképeknél ezért ezen felismerési folyamat nem kerül tárgyalásra. Fontos megemlíteni, hogy a kataszteri térképek esetében ez a felismerési folyamat a nevek, adatok tekintetében jóval egyszerűbb (azonos betűtípus és betűméret, egy azon szó esetében pedig azonos állású betűk), így ebben a témában a kutatások révén [19, 12] jó eredményeket sikerült elérni.

6.2. Tudás alapú megközelítés

A geoinformatika egyik fontos feladata a rendelkezésre álló raszteres térképek vektoros modellé konvertálása. Manapság ezt főként kézzel végzik, a poligonok körberajzolásával. A használatban lévő szoftverek kényelmesebbé teszik a munka elvégzését, azonban nem foglalkoznak a térkép értelmezésével. A térinformatikában a térkép értelmezése bonyolult feladat. Ennek a feladatnak a megoldásához különböző képfeldolgozó algoritmusok szükségesek. Az algoritmusok végrehajtásának sorrendjét sokszor a számítógépes értelmezés logikája határozza meg. Cél, hogy a felhasználó szakértői munkájának nagy részét a topografikus térképek esetében átvállaljuk és olyan rendszert adjunk (lásd 32. ábra), amelybe a felhasználó bármikor beavatkozhat.

Ebben a rendszerben a jelkulcsi elemek az őket felismerő algoritmusok szerint három csoportba sorolhatók. Megkülönböztetünk *pontszerű* jelkulcsi elemeket (helyhez kötött objektumok jelzésére, lásd 31. ábra *a* *c*, illetve egy felszínborítás ábrázolására, pl. szőlő



32. ábra. A topográfiai térkép-interpretáló rendszer folyamat ábrája.

szimbólum), *vonalas* jelkulcsi elemeket (pl. szaggatott vonal, kettős vonal, szimbolikus vasútvonal, lásd 31. ábra *d f*), valamint *textúrázott felület alapú* jelkulcsi elemeket (adott felület típust lefedő mintázat, lásd 31. ábra *g i*). A fent említett jelkulcsi elemeken túl a területeket körülhatároló poligonok felismerésével külön kell foglalkoznunk.

Egyes térképi objektumok ábrázolása több osztályra is kiterjed, pl. egy tavat alapvetően egy homogén kék felület ábrázol, de a határa egy vonalas elem. Szintén érdekes kérdés a házak poligonjának felismerése, ahol a felület kis méretéből adódóan a határolóvonal tekinthető meghatározó tulajdonságnak.

A folyók felismerése egyike a legnehezebb problémáknak, mivel a folyó szélességének ábrázolása a vonalas és a felületi kategória között változhat (pl. ha a folyó delta torkolattal rendelkezik). Az alábbi táblázatban (lásd 1. táblázat) néhány jellegzetes térképi objektum besorolása látható.

Objektum	Pontszerű	Vonalas	Textúrázott felület
Betű	X	–	–
Szőlő szimbólum	X	–	–
Út	–	X	–
Vasút	–	X	–
Ház poligon	–	X*	–
Erdő	–	X*	X
Folyó	–	X*	X
Folyó delta	–	X*	X
Mező	–	X*	X
Tó	–	X*	X

1. táblázat. X - Felismerhető típus, * - Poligonhatár típusként is felismerhető.

A későbbi fejezetekben a különböző jelkulcsi elemek felismeréséhez tartozó tudásalapú algoritmusok kerülnek bemutatásra.

Ahhoz, hogy ezekből az eljárásokból egy térképi interpretációs rendszert készülhessen, ismerni kell a térkép elkészítése során alkalmazott sorrendiséget, továbbá figyelembe

kell venni a térképet olvasó ember gondolkodásmódját. Első lépésként a térkép összeállítására vonatkozó szabályokat tekintjük át röviden, amelyek egy-egy réteget hoznak létre.

1. Megrajzoljuk a térképi poligonok határvonalait.
2. A poligonokat kifestjük, vagy textúrával fedjük le.
3. Megrajzoljuk a vasút és úthálózatot, a jelkulcsi elemeiknek megfelelő reprezentációval.
4. Rárajzoljuk a térképre a pontszerű objektumok jelkulcsi elemeit.
5. A térképre írjuk a pontszerű objektumokhoz tartozó szöveges adatokat.
6. A vonalas objektumokhoz tartozó feliratokat vagy a vonal mellé írjuk, vagy a vonalat a szöveg helyén megszakítva a vonal helyére, de mindkét esetben a vonal ívét követve.

A vektorizálás során a térképkészítésnél ismertetett lépéseket fordított sorrendben alkalmazva próbáljuk a térkép komponenseit legyűjteni. Látható, hogy a térképi rétegek egymásra rajzolódnak, a felsőbb rétegek az alattuk levőkből pedig kitakarhatnak részeket. A térképészetben a vázolt szabályrendszer összetettebb, így rétegen belül is felléphet interferencia, pl. egymást átfedő feliratok esetén a szöveges adatokból elhagyunk, hogy ne írodjanak egymásra (térképi generalizálás).

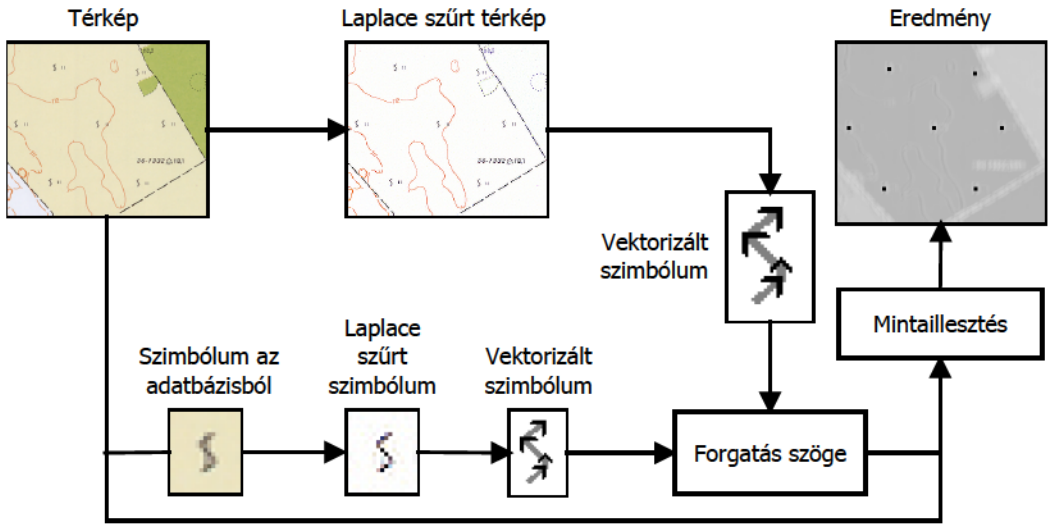
6.3. Pontszerű jelkulcsi elemek felismerése

A pontszerű jelkulcsi elemek kisméretű ábrák (lásd 31. ábra *a* *c*), amelyek megtalálhatók a jelmagyarázatok gyűjteményében. Az adott térképen szereplő jelkulcsi elemek általában megtalálhatók a térkép alján feltüntetett jelmagyarázatban. A térképen mindig torzítatlan formában szerepelnek, néha elforgatott helyzetben.

A [21, 22, 25] cikkeimben részletesen bemutatott hatékony algoritmus képes pontszerű jelkulcsi elemek lineáris idejű felismerésére (az algoritmust felhasználva nem csak a pontszerű szimbólumokat, hanem a felületi textúrákat is felismerhetjük, mivel a textúra egysége pontszerű szimbólumhoz hasonló képi elem).

Cél a térképen szereplő minden pontszerű jelkulcsi elem felismerése. A program jelenleg az adatbázisban tárolt összes jelkulcsi elemre megkísérli a felismerést, illetve a felhasználó szűkítheti ezeknek a körét a ténylegesen szóba jövő jelkulcsi elemekre. Az algoritmus működési elve egy jelkulcsi elemre (lásd 33. ábra) fentebb látható, a szóba jövő jelkulcsi elemekre pedig a felismerés a következő lépések elvégzését jelenti.

1. Hajtsunk végre élszűrést (pl. Canny vagy Laplace) a teljes térképen és Otsu küszöböljük az eredményt.
2. A szűrt, csak éleket tartalmazó térképet (vékonyító szűrő alkalmazása után) vektorizáljuk.



33. ábra. Egy tetszőleges szimbólum detektálásához tartozó felismerési folyamat. A jobb láthatóság miatt az eredményképnek és a digitálisan szűrt képnek a negatívja került feltűntetésre. Az eredményképen látható egyes fekete pontok a felismert szimbólumok helyeit jelentik.

3. Minden szóba jövő jelkulcsi elemre hajtsuk végre a következő lépéseket:

4. Hajtsunk végre élszűrést és Otsu küszöbölést a külön álló, $s_x * s_y$ pixel méretű jelkulcsi elemen.

5. Vektorizáljuk a jelkulcsi elem szűrt képét.

6. A transzformált jelkulcsi elem képén sorfolytonosan keressük meg az első $s(u, v)$ pixelt. Határozzuk meg ezen élpixelhez tartozó d_s vektor irányát.

7. Az élszűrt térkép minden egyes $m(x, y)$ élpixelére határozzuk meg a d_m vektor irányát és hajtsuk végre a következő lépéseket:

8. Forgassuk el az eredeti jelkulcsi elem raszteres képét az $s(u, v)$ pont körül a d_m és d_s vektorok által bezárt szöggel.

9. Illesszük az elforgatott jelkulcsi elemet az $s(u, v)$ pontjával a térkép $m(x, y)$ pontjára.

10. Tekintsük ezután az eredeti jelkulcsi elemet reprezentáló elforgatott raszteres kép S_{mat} mátrix reprezentációját. A mátrix elemeiből vonjuk ki az adott pont alá eső térképi pixel értékét.

11. Számítsuk ki az S_{mat} különbség mátrix σ szórását.

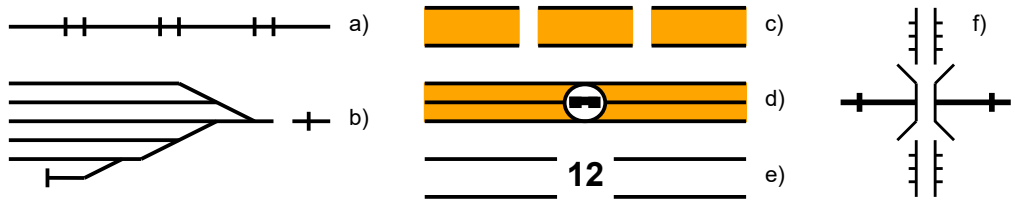
12. Ha a σ szórás kisebb, mint egy adott küszöbérték, akkor a jelkulcsi elemet felismertük. Ennek helye pedig $(x \quad u+s_x/2, y \quad v+s_y/2)$.

A fenti módszer tovább is gyorsítható. Ehhez vezessünk be egy segédmátrixot. A mátrix adott elemének koordinátája legyen (x, y), értéke pedig az élszűrt kép $(0, 0)$ és (x, y) pontjai közé eső területen lévő élpixelek száma. A segédmátrix elemei lineáris futási időben kiszámíthatóak (lásd a Digitális szűrők fejezet Kis előfordulási pixelek eltávolítása alfejezetben adott módszert). A mátrix segítségével konstans időben (négy művelettel) kiszámítható az élszűrt és Otsu küszöbölt térkép bármely szimbólum méretű területére, hogy az hány élpixelt tartalmaz. Az adott területen előfordulű élpixelek száma kizáró feltételként használható, ha az túlságosan eltér az élszűrt és Otsu küszöbölt szimbólum élpixeleinek számától.

A módszer összevetésre került a [27] módszerrel, ami kb. 95%-os felismerési aránnyal rendelkezik. A szerző által adott módszer napjaink átlag gépén nagyjából 261 másodperc alatt képes egy 100 megapixeles térkép feldolgozására, míg az új módszernél ez 2-3 másodpercet vesz igénybe. Említésképpen, a pontszerű szimbólumok manuális vektorizálása akár 1 óráig is eltarthat. A módszert jellemző lineáris futási idő a [21] cikkben került bizonyításra.

6.4. Vonalas jelkulcsi elemek felismerése

A vonalas jelkulcsi elemek esetén útszerű objektum nyomvonalára, illetve poligon objektum határvonalára rajzolódik rá egy meghatározott struktúrájú vonalas attribútum (lásd 34. ábra).



34. ábra. Példák vonalas jelkulcsi elemekre: a) közúti villamosvasutak, b) vasúthálózat egy vasútállomáson, c) épülő műút, d) autópálya, (autósztráda), út menti segélykérő telefon, e) út, szélesség megírással, f) csatornán átívelő híd.

A vonalas jelkulcsi elemek felismerése a raszter-vektor konverzió egyik legnehezebb feladata. Ezek a jelkulcsi elemek gyakran összetett, bonyolult ábrázolásmóddal rendelkeznek. Megengedett, hogy két jelkulcsi elem csak méretarányában térjen el egymástól, valamint az is, hogy a vonalas jelkulcsi elemek keresztezzék egymást vagy összekapcsolódjanak (lásd 34. ábra a – b). Szintén nehézséget okoznak az összetartozó, párhuzamosan futó vonalak (lásd 34. ábra c – f, amelyek megkülönböztetése a külön objektumokhoz tartozó, párhuzamosan futó vonalaktól még gyakorlott szemmel is nehézkes lehet. Gondot okoznak továbbá a szaggatott jelkulcsi elemek is (lásd 34. ábra b).

Az imént felsorolt anomáliák felismerése túlmutat ennek a disszertációnak a célkitűzésein. Az ilyen esetektől eltekintve bemutatok egy a [24] cikkemben már ismertetett topológiai alapon működő vonalas jelkulcsi elem felismerő módszert. Ehhez feltesszük, hogy a jelkulcsi elemek

1. nem keresztezik egymást,
2. nem találkoznak és nem alkotnak egyetlen objektumot, továbbá
3. folytonosak, azaz összefüggőek.

A vonalas jelkulcsi elemek vektorizálát mutatja be a [11] cikk, ahol bináris, raszteres képformátumú kataszteri térképek kerülnek vektorizálásra. A színes topografikus térképek elemeinek további tulajdonságai, mint út szélesség, burkolattípus stb., azonban nem ismerhető fel a klasszikus módon [14]. Ezen tulajdonságok leírásához szükség lehet reprezentációs szinten a színre, struktúrára és további megfelelő jellemzőkre. A következő módszer képes felismerni a vonalas jelkulcsi elemek nyomvonalát egy topográfiai térképen:

1. Hajtsunk végre szegmentálást a raszteres térképen és soroljuk osztályokba a pixeleket.
2. Készítsünk egy bináris térképet, melynek a fekete pixelei pl. úthoz tartoznak.
3. Alkalmazzuk a vékonyító szűrőt, majd pedig a morfológiai vékonyító szűrőt a binárizált térképen.
4. Vektorizáljuk az egy pixel vastagságú drótvázat.

Az első lépésben a szűrőket tárgyaló fejezetben bemutatott réteg-alapú színdetektálás alkalmazására kerül sor. Ezután egyszerű maszkolás segítségével kiválasztjuk a kellő színeket. Ezekből lesznek a fekete pixelek (vonalas objektumhoz tartozó), a többi pedig fehér pixel lesz. A módszer harmadik lépését a szűrőknél már szintén részletesen bemutatott morfológiai szűrőkkel lehet elvégezni.

	1	
1	1	1

	1	
	1	1

2. táblázat. Bináris képeken történő morfológiai elágazás-detektáláshoz használatos struktúráló elemek. Az elemek értékei: 0 - háttér, 1 - objektum. Az üres helyeken lévő értékek tetszőlegesek lehetnek.

A drótvázzá alakított bináris képet a következő módon vektorizálhatjuk. Fessük az összes objektumhoz tartozó pixelt *feketére*, míg a felülethez tartozóak *fehérre*. Legyen

ezután *piros* az összes olyan *fekete* pixel, melyeknek legalább három *fekete* szomszédjuk van, azaz elágazási pontok. A megmaradt *fekete* pixelekre alkalmazzuk a fenti (lásd 2. táblázat) morfológiai struktúráló elemeket. Ez azt jelenti, hogy a vizsgált pixelre és környezetére illesztjük a struktúráló elemeket és azok 90•-okkal elforgatott változatait (lásd morfológiai szűrők működésének leírása a szűrők fejezetben). Egyezés esetén, azaz sikeres detektálásnál a megfelelő pixelt *kékre* színezzük. A *piros* elágazási pontok vonalakat kötnek össze, míg a *kék* pontok elágazásokat. A fennmaradó *fekete* pixelekből jelöljük *zölddel* azokat, melyeknek legfeljebb egy szomszédjuk fekete. Ezek jelentik a vonalak végződéseit, akár valóst, akár az elágazásnál történő végződést. Látható, hogy a színeken a következő prioritás definiálható: *fehér* < *fekete* < *zöld* < *piros* < *kék*. A következő lépések a színezés segítségével vektorizálják az objektumokhoz tartozó pixeleket

1. Válasszunk ki egy *zöld* pontot és fessük *fehérre*, majd készítsünk egy új vonalszegmens listát és adjuk hozzá a listához ezt a pontot.

2. *Fekete* aktuális pont esetén válasszunk egy *fekete* szomszédot ha van. Egyébként válasszunk egy nagyobb prioritású pontot a szomszédok közül. Legyen a pont *fehér* és adjuk a lista végéhez.

3. Ugorjunk a 2. lépéshez, amíg van megfelelő szomszéd.

4. Menjünk vissza a list első elemének helyéhez és ugorjunk a 2. lépésre. Fontos, hogy innentől kezdve az új pontokat a lista elejére kell majd beszúrni. (Ez a lépés a pontokat az ellenkező irányban elindulva kezdi feldolgozni.)

5. Ugorjunk az 1. lépésre, amíg létezik *zöld* pont.

6. Válasszunk egy *fekete* pontot, majd színezzük *fehérre*, továbbá hozzunk létre egy új vonalszegmens listát, amely tartalmazza ezt a pontot. Legyen ezután ez a lista az aktuálisan karbantartott.

7. Válasszunk az aktuális pont szomszédai közül egy *feketét*, jelöljük meg *fehérnek*, és tegyük a lista végére.

8. Ugorjunk a 7. lépésre, amíg létezik *fekete* színű szomszéd.

9. Válasszunk egy *piros p* pontot, színezzük *fehérre*, majd hozzunk létre egy új vonalszegmens listát, mely tartalmazza. Legyen
   Szomszéd = *Piros* = *Számláló* ↑ 0,
   Hol ↑ **utolsó**,
   q ↑ *p*.

10. *MegelőzőPont* ↑ *q*.

11. Ha a *Szomszéd*-adik *r* szomszédja *q*-nak létezik, akkor
    q ↑ *r*,

$n \uparrow q$,
Szomszéd $\uparrow$ *Szomszéd* + 1.
Tegyük q-t a lista *hol* helyére, majd ugorjunk a 13 lépésre.

12. Ha a *Piros*-adik r szomszédja q-nak létezik,

    (a) Ha q és n szomszédok és *hol* = **első**, akkor $q \uparrow$ *MegelőzőPont* és *Piros* $\uparrow$ *Piros* + 1. Ugorjunk a 10. lépésre.

    (b) Tegyük q-t a lista *hol* helyére, és legyen q színe *fehér*, valamint *Szomszéd* $\uparrow$ 0 és *Számláló* $\uparrow$ *Számláló* + 1. Ugorjunk a 10. lépésre.

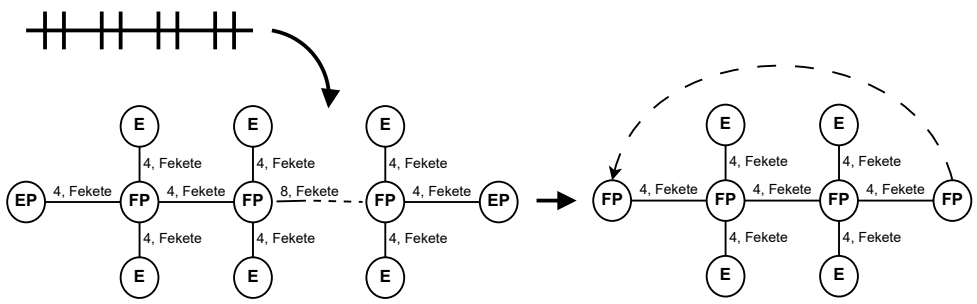
13. Ha *hol*=**utolsó**, akkor *hol* $\uparrow$ **első**, $q \uparrow p$ és ugorjunk a 10. lépésre.

14. Ugorjunk a 9. lépésre, amíg létezik *piros* pont.

A fenti eljárás után két tulajdonságot tudunk felismerni a vektorizált jelkulcsi elemen: elágazást ($F \ll Fork$), illetve végpontot ($E \ll End$). Ezek az adatok jól ismertek az ujjlenyomat felismerő rendszerekben, ahol *minutiae* pontoknak hívják őket. Mivel az ujjlenyomat esetében egy mintázat elágazása a komplementer mintában egy végződést jelent, ezért a számítógépes gyakorlatban csak az egyiket használják azonosításra. Esetünkben mindkét jellemzőt használnunk kell, mivel nem beszélhetünk komplementer mintázatról. A vonalas jelkulcsi elemek reprezentálására az irányítás nélküli gráfokat használjuk. Egy jelkulcsi elemnek az ún. EF-gráfja a következőképpen definiálható:

- A csúcsok típusa E vagy F lehet A csúcsok színét a vektorizálás előtti, a binarizálást megelőző színük adja.
- Két csúcs között akkor vezet él, ha a két pontot összekötő vonalszegmens sorozat nem tartalmaz csúcsot. Az él súlya megegyezik a két csúcsot összekötő út hosszával, a felhasználó pedig további éleket hozhat létre a különböző színű csúcsok összekötésére.
- Legyenek továbbá különleges P indexű csúcsok azok, amelyek a vonalas objektum útvonalán vannak. (Ezek a végső vektoros modell előállításában lesznek fontosak.)

Természetesen EF-gráfot nem csak jelkulcsi elemhez, hanem a térkép nyers vektorizált változatához is megfeleltethetünk (ahol minden vonalas elem még egyetlen tagolatlan egységet alkot). A térképi felismerésben a vonalas jelkulcsi elemek legkisebb elemi egységeit használjuk. Az elemi EF gráf a jelkulcsi elem azon legkisebb egységét ábrázolja, amelynek iterálásával a teljes jelkulcs előállítható. Az elemi EF gráfban általában két olyan csúcs szerepel, amelyek ebben az iterált kapcsolódásban részt vesznek. Ezeknek a típusa F lesz, azonban csak egy éllel rendelkeznek. Ezek a gráf ki, illetve belépési pontjai. Ritkán találkozunk olyan jelkulcsi elemmel, ahol nem azonosítható az elemi egység ki és belépési pontja. Ekkor a jelkulcsi elem egysége önmaga. A térkép nyers vektorizált változatában nincs értelme a legkisebb elemi egység definiálásának.



35. ábra. Egy többvágányú vasútvonal EF gráfja és elemi EF gráfja. A feltüntetett távolságértékek egymáshoz viszonyított, relatív értékek, melyet a raszteres térkép léptékéhez és felbontásához aktualizálni kell a feldolgozásakor. A szaggatott vonal az elemi EF gráf ciklikus tulajdonságát reprezentálja.

A 35. ábra mutatja, hogy egy jelkulcsi elem az elemi EF gráfjának többszörözésével felépíthető. A példán látható jelkulcsi elem gráfjában szereplő élsúlyok egymáshoz viszonyított arányszámok, és a térkép méretarányának függvényében használatosak. A súlyok mellett még színt is rendelünk az egyes élekhez (pl. a 35. ábra minden él színe fekete). A jelkulcsi elemekhez tartozó elemi EF-gráfokat célszerű csúcsmátrixal reprezentálni csekély méretüket tekintve.

A térképi vonalas objektumok azonosítását részgráf keresési problémára vezethető vissza: a térkép nyers vektoros modelljének EF gráfjában keressük a jelkulcsi elemekhez tartozó elemi EF gráfok előfordulásait. Ehhez először normalizáljuk a térképi méretarányhoz a jelkulcsi elemek elemi EF-gráfjainak gyűjteményét. Rendezzük továbbá a normalizált EF gráfok gyűjteményét a maximális fokszámú csúcsaik szerint csökkenően és hívjuk ezt a gyűjteményt S-nek.

A fenti feltételek alapján készíthető olyan algoritmus, amely részgráfok felismerésére használható. A térkép feldolgozása során felismert jelkulcsi elemeket eltávolítjuk oly módon, hogy a megfelelő részgráfot átszínezzük. Két féle színt használhatunk, például kéket és pirosat, hogy nyomon kövessük az algoritmus folyamatát és megállapíthassuk mikor szükséges terminálni azt. Kezdetben minden gráfcsúcs legyen piros. A következő algoritmus akkor áll meg, ha már nem található további piros csúcs a gráfban.

1. Válasszuk ki a térkép EF gráfjának egy maximális fokszámú piros csúcsát.

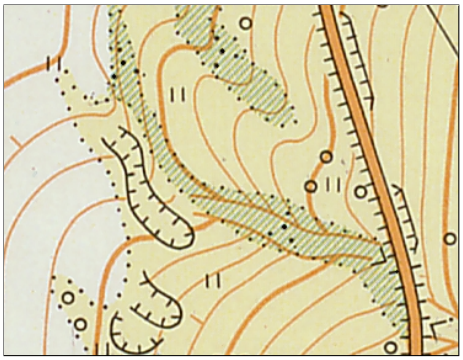
2. A jelkulcsi elemek EF gráfjainak rendezett sorából az egyező maximális fokszámúakat sorban megpróbáljuk illeszteni, az első sikeres felismerésig:

   (a) Az elemi EF gráf minden maximális fokszámú csúcsára próbáljuk ki az illesztést az első sikeres találatig, az alábbi módon:

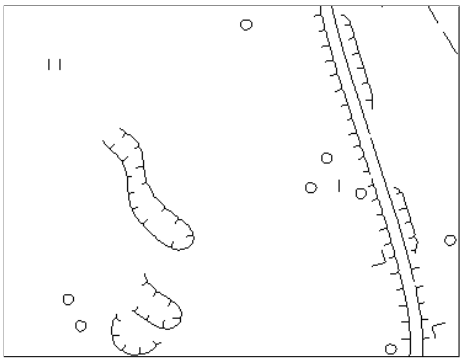
      i. Az EF gráf és az elemi EF gráf illesztett csúcsaiból kiindulva hajtsunk végre egy párhuzamos szélességi bejárást. Ezt akkor mondjuk sikeresnek, ha az elemi EF gráf teljesen, azaz a belső csúcsok fokszámát és az élhosszakat illetően megegyezően fedi az EF gráf egy részét.

ii. Ha a két struktúra illesztése sikeres, akkor az EF gráf illesztett részét színezzük kékre.

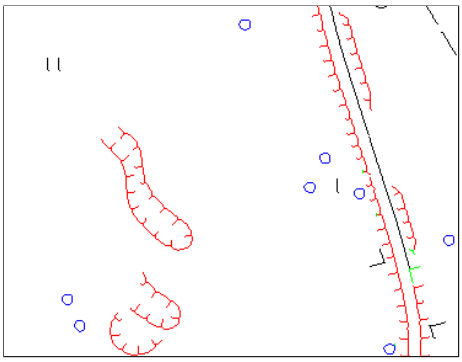
3. Ha az előző lépések során nem történt színezés, akkor kilépünk, egyébként ugorjunk az 1. lépésre.



(a) Eredeti térképrészlet



(b) Előszűrt, vékonyított adatok



(c) Végeredmény

36. ábra. Példa egy térképrészlet vonalas adatainak EF gráf alapú feldolgozására. A képen a forráskép, a belőle előszűréssel és vékonyítással előállított köztes raszteres kép, valamint végül az EF pontok és struktúrák vektoros adatokból történő felismerése látható.

A felismert elemi EF-részgráfot úgy távolítjuk el a térkép EF gráfjából, hogy a ki-, illetve belépési pontjait csak akkor töröljük, amennyiben azok E csúcsok. Törlés esetén a részgráfban érintett, de a gráfban maradó F csúcsok fokszámát csökkentenünk kell. Az élek hosszának egyezését közelítőleg értjük, a vonalas jelkulcsi elemek lehetséges görbületeinek következményeként. A módszer eredményét a mellékelt ábra mutatja (lásd 36. ábra).

6.5. Homogén felületek felismerése

A térképen található felületek felismerése során gyakran előforduló tevékenység a vizuálisan elválasztott, valójában összetartozó felületek egyesítése, valamint a térképi hibák

kijavítása. A felületek felismerését egy olyan maszk segítségével valósítjuk meg, amely megadja, hogy a térkép mely pontjai tekintendők felületi pixeleknek. A maszk megadható például a pontszerű és vonalas jelkulcsi elemek eltávolítása után keletkező területtel. További lehetséges módszer az, hogy a térképen közvetlenül elvégzett színszegmentálás után a felületi színekkel rendelkező pontokat tekintjük maszknak. A maszkhoz tartozó pontokban a megfelelő felületi szín indexét állítjuk be, a maszkon kívüli pontokat pedig *NEM DEFINIÁLT* típusúnak tekintjük. A megfelelő eredmény érdekében az említett két módszert együttesen alkalmazzuk. Ahhoz, hogy a felületeket felismerjük és jó minőségben poligonizálhassuk, egy heurisztikus szabályrendszert készítettem. A továbbiakban sorra bemutatom a felismeréshez használt szabályokat.

Ál-felületek pixeleinek eltávolítása

A szabály értelmében eltávolítjuk azokat az adott felület-típusú pixeleket, pixelcsoportokat, amelyek előfordulási száma egy megadott küszöbértéknél kisebb. Így eltávolíthajtuk az esetleges félreosztályozással létrejövő ál-felületeket. Az alábbi táblázatban (lásd 3. táblázat) található példa a fű besorolásához szükséges szabályt adja meg.

Felület típusa	**Keresési terület mérete**	**Min. pixel előfordulás**
Fű	5x5 pixel	13 darab

3. táblázat. Példa fűvel borított felülethez tartozó szabályra.

Egy képpontot pontosan akkor sorolunk a fű felszíni borításhoz, ha körülötte az $5 * 5$-ös négyzetben legalább 13 olyan képpont található, amely fűként értelmezhető. A kép szélein, ahol nincs $5 * 5$-ös befoglaló négyzet a pont körül, a küszöbértéket arányosan csökkentjük. A szabályt alkalmazása balról-jobbra, sorfolytonosan történik. Eljárásunk nem adaptív abban az értelemben, hogy a maszk új képként alakul ki, az input változatlan marad.

Határolt felületek osztályozása

A topográfiai térképeken gyakori az, hogy bizonyos felületeket körülrajzolnak. Ez nagyban segíti a felismerést, ha tudjuk, hogy a megrajzolt poligon kizárólag egy adott típusú, homogén felületet határol. Ilyenkor, ha találunk egy domináns felület típust a határoló vonalon belül, akkor a belső területet besorolhatjuk a megtalált típushoz. Például a ház esetén alkalmazott rózsaszín kizárólagos és mindig fekete határvonallal párosul. A körülhatároltság vizsgálatához az „elárasztás” módszerét alkalmazzuk az adott típusú felület elem bármely pontjából. Az ennek során érintett képpontokat a felület típusának megfelelően színezzük át. A gyakorlatban a határoló vonalak megszakadhatnak, ezért célszerű egy küszöbértéket adni az elárasztás kiterjedésére. Például ház esetén (lásd 4. táblázat) ez a határ 500 pixel lehet.

Határoló	Blob típusa	Elárasztás megengedett mértéke
Fekete	Ház	500 pixel

4. táblázat. Példa házak azonosítására szolgáló szabályra.

Ha az adott korláton belül az elárasztás nem áll meg határvonalnál, akkor leállítjuk az eljárást és a vizsgált területet *NEM DEFINIÁLT* státuszúra állítjuk.

Összetartozó területek összevonása

A számítógépes interpretálás során a felületekre később rárajzolt vonalas és pontszerű jelkulcsi elemeket eltávolítjuk, ezáltal felületi szakadások jönnek létre. A kettévágott területek összevonására külön szabályt kell alkalmaznunk (lásd 5. táblázat). A szabályban azt rögzítjük, hogy amennyiben a maszk sorfolytonos feldolgozása során nem a várakozásnak megfelelő pixelt találunk, akkor hogyan kell eljárnunk. Az adott pontból kiindulva két ellentétes irányba terjeszkedve azonos típusú felületet keresünk. A keresést a sugárirány körbeforgatásával végezzük, és egy adott távolságon belül maradunk. Ha sikerrel jártunk, akkor a kiindulási ponton átszínezzük a felület pontjává.

Nem ritkán előfordul, hogy a felület szakadásának egy pontjából nem jutunk el a felület jellemző pontjaihoz az adott korláton belül. A teljes térkép hiánypótlása után azonban már az ilyen izolált pontokban is nő a javítás esélye. Ezért a szabályt célszerű többször végrehajtanunk a teljes térképen.

Blob típusa	Keresési sugár	Iterációk száma
Fű	4 pixel	4

5. táblázat. Példa a fű típusú felületen lévő lukak eltávolításához tartozó szabályra.

Füves terület esetén például elegendőnek bizonyult maximum $2 * 4$ pixel szélességű szakadást feltételezni, és $4*$ végrehajtani a hiányok feltöltésének algoritmusát.

Kis méretű hibás felületek és lyukak eltávolítása

A térképen gyakran előfordulnak kicsiny kiterjedésű hibák. Egy jól definiált felületen belül keletkezhet látszólag nem odatartozó folt. Például erdős területen belül olykor hiány jelenik meg, de az interpretálás során ezt a területet is erdőként kell azonosítani. Másrészt, az is előfordul, hogy egy kis kiterjedésű terület típusa azonosíthatónak tűnik, azonban csekély mérete és tőle különböző, de nem karakterisztikus környezete miatt ez nem fogadható el. Ilyenkor a pontot eltávolítjuk.
A két esetet magában foglaló szabályt a 6. táblázat tartalmazza.

Blob típus	Határoló típus	Max. blob méret	Döntés
Nem-erdő	Erdő	500px	*Típus:=Erdő*
Erdő	Nem-erdő	250px	Törlés

6. táblázat. Blobok és lukak törlése az erdő térképi rétegről

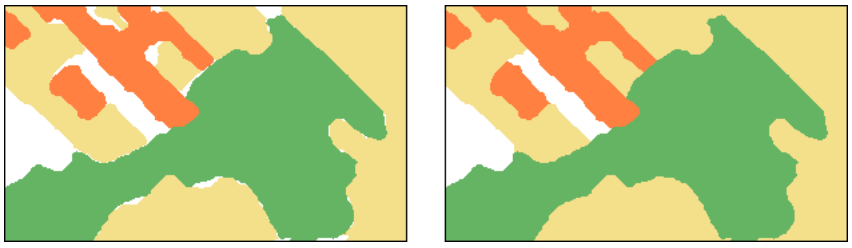
Különböző felületekkel határolt ismeretlen terület azonosítása

Eddigi szabályaink alapján nem tudjuk besorolni a térkép olyan osztályozatlan pontjait, amelynek közelében több különböző típusú felület található. Ha az azonosítatlan terület elég kicsi, vagy a besorolatlan ponthoz elég közel jól meghatározott felületet találunk, akkor az osztályozás elvégezhető. A felderített környezet alapján a megfelelő szabály megadja a domináns környezeti típust, amelyre a pontot vagy pontokat átszínezzük.

Típus	Metódus	Domináns típus	Al-domináns típus	Max. távolság	Max. méret
Ismeretlen	Blob	Fű	{Erdő, Szőlő}	10	100
Ismeretlen	Blob	Szőlő	{Erdő}	15	200

7. táblázat. Különböző típusú objektumok által határolt blobok kitöltése

A fentebbi táblázatban (lásd 7. táblázat) két olyan szabályt látunk, amely füves terület, erdő és szőlő, illetve szőlő és erdő által közrezárt ismeretlen képpontokat sorol be a domináns típusba. A szabályok alkalmazását mutatja be a 37. ábra.



37. ábra. Többféle blobbal határolt blobok, pixelek és lukak.

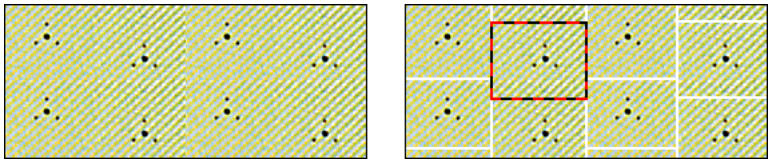
Felület kiválasztása, térképi felületrétegek egyesítése

Az előzőekben leírt szabályok körültekintő alkalmazását támogatja az adott típusú felületek kiválasztását végző szabály. Ha a szabály alkalmazás célfelületeit gondosan kiválasztjuk, a többit pedig kimaszkoljuk, akkor nem lépnek fel nem kívánt mellékhatások.

Ha az egyes szabályokat a térkép kimaszkolt változataira hajtjuk végre, akkor szükség van az eredmények együttes megjelenítésére egyetlen térképen. Ezt a lépést támogatja a felületrétegek egyesítésének szabálya.

6.6. Textúrázott felületek felismerése

A textúrázott felületekre jellemző, hogy rendelkeznek egy olyan kis ábrával amely mozaikszerűen lefedi őket. A felületek textúrájának állása mindig rögzített. A pontszerű jelkulcsi elemek felismerésénél bemutatott eljárást kis méretű ábrákhoz használtuk. Ezt az algoritmust általánosíthatjuk úgy, hogy alkalmas legyen a textúrázott felületek felismerésére (lásd 38. ábra). Ebben az esetben az algoritmus forgatásra vonatkozó lépései elhagyhatóak.



38. ábra. Bokrokkal borított területet reprezentáló textúra. a) textúra, b) résztextúrákkal mozaikolt textúra, amely a kernelt mutatja.

Feltesszük, hogy rendelkezésre állnak – a pontszerű jelkulcsi elemek mellett – a térkép méretarányához tartozó felületi textúrák raszteres adatai is. Feltesszük továbbá, hogy az egyes textúra típusok indexükkel azonosíthatók. A 4. fejezetben bemutatott algoritmus pontszerű jelkulcsi elemek helyett most textúra kernelek illesztését végzi adott statisztikai küszöb mellett. A textúrázott felületeket a következő módon kíséreljük meg felismerni.

1. Készítsünk a térkép méretével megegyező méretű bináris szimbólum-maszkot, amelyben az összes érték kezdetben *HAMIS*.

2. Készítsünk a térkép méretével megegyező méretű felület-maszkot, amelynek minden pontja egy index lesz. Ez a térképen az adott helyen lévő felület típusának indexe lehet, vagy *NEM DEFINIÁLT*. Kezdetben az összes értéket *NEM DEFINIÁLT*-ra állítjuk.

3. Hajtsunk végre élszűrést a térképen (pl. Laplace).

4. Rangsoroljuk a textúra kerneleket szórásuk nagysága szerint.

5. Dolgozzuk fel a térkép még nem maszkolt él pixeleit sorfolytonosan

   (a) A textúra kernelek rangsor tömbjének minden eleméhez *IGAZ* értéket rendelünk (még nem teszteltük egyezésre).

   (b) Bináris kereséssel kikeressük a textúra kernelek közül az aktuális térképi részlet szórásához legközelebb álló szórásút, amelyhez *IGAZ* érték van rendelve.

   (c) Végrehajtjuk az aktuális textúra kernel illesztését a megengedett szórási eltérés mellett.

(d) Felismert textúra esetén a felület-maszkon a lefedett terület pontjaihoz rendeljük hozzá a textúra típusának indexét. A szimbólum maszk megfelelő pontjait pedig *IGAZ*-ra állítjuk.

(e) Ha nem történt felismerés, akkor a rangsorban *HAMIS*-ként jelöljük meg a textúra kernelt és ugorjunk a b) lépésre.

A textúrázott felületek határvonala mentén általában a textúra kernelnek csak a részei fordulnak elő. Ezek felismerésére a fenti algoritmus nem alkalmas. Egy ilyen töredék rész felismerését a szomszédos, azonosított textúra téglalapok alapján pontonként végezhetjük.

7. Waveletek a térképi interpretációban

A térképi interpretáció koncepciójának kialakítása során megvizsgáltam a wavelet alapú szűrők, illetve erre épülő eljárások alkalmazhatóságát. Bár nem kerültek később bele a végleges modellbe, mégis tanúlságos volt ezek vizsgálata. Alternatív lehetőséget mutattak a pontszerű jelkulcsok felismerésére, az interpretációs tudás kibővítésére. A következőkben több, a waveletekre épülő megközelítést fogok bemutatni.

7.1. Pontszerű jelkulcsok alternatív felismerési módja

A pontszerű jelkulcsok felismerésénél egy olyan módszert mutattam be, amely forgatásinvariáns mintaillesztést képes végrehajtani lineáris időben (viszonylag kis konstanssal), kihasználva a jelkulcs és a térkép speciális tulajdonságait. A pontszerű jelkulcsi elemeket objektumként vizsgálva egy másik, az irodalomban jól ismert detektálási módszer [29] is kínálkozik a feladat megoldására. Ezt leginkább arc, rendszámtábla, karakter stb. detektálására használják.

Viola és Jones a *Haar-like feature*-ként elnevezett képrészlet tulajdonságokból indul ki. A módszer egyik kulcsfontosságú alapja az *integral image* használata. A szürkeárnyalatos képek esetében ez egy olyan mátrix létrehozását jelenti, amelynek dimenziói megegyeznek a forrásképével. Ekkor a mátrix adott elemének értéke megegyezik a kép ugyanazon helye és a kép bal felső sarka által közrezárt területen található intenzitásértékek összegével:

$$I(i,j) = \sum_{(a,b)=(0,0)}^{(i,j)} f(a,b).$$

A mátrix értékeinek kiszámításáról könnyen belátható, hogy minden egyes eleme konstans 1 művelettel előállítható az alábbi módon:

$$\leq I^{\circ}(0,0) = \chi(f(0,0) = ind)$$

$$\leq I^{\circ}(i,0) = I^{\circ}(i \quad 1,0) + \chi(f(i,0) = ind)$$

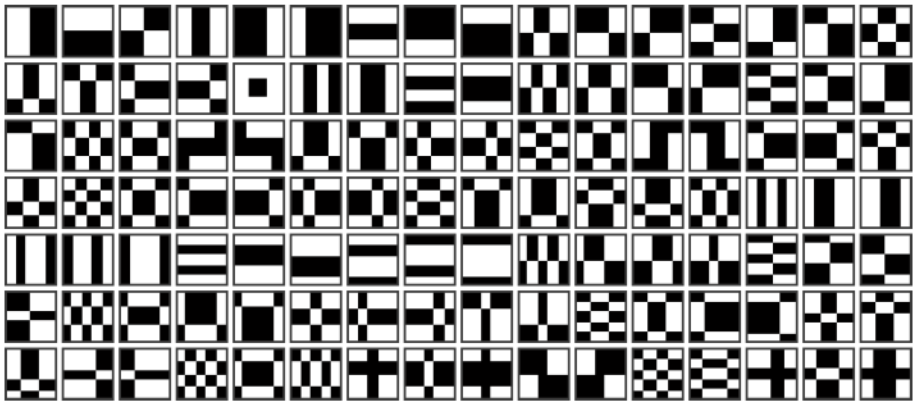
$$\leq I^{\circ}(0,j) = I^{\circ}(0,j \quad 1) + \chi(f(0,j) = ind)$$

$$\leq I^{\circ}(i,j) = I^{\circ}(i \quad 1,j) + I^{\circ}(i,j \quad 1) \quad I^{\circ}(i \quad 1,j \quad 1) + \chi(f(i,j) = ind), \text{ ha } i > 0 \text{ és } j > 0.$$

Viola és Jones ennek a speciális mátrixnak a segítségével elérte, hogy a kép bármely $k * l$ méretű téglalap alakú részletére konstans időben számíthatjuk ki az alatta lévő intenzitásértékek összegét az alábbi módon:

$$db = I^{\circ}(i,j) \quad I^{\circ}(i \quad k,j) \quad I^{\circ}(i,j \quad l) + I^{\circ}(i \quad k,j \quad l).$$

Ennek abban volt nagy jelentősége és újdonsága, hogy ezek segítségével konstans időben tudták ezután kiszámítani a kép tetszőleges méretű részletének haar-szerű tulajdonságát. A haar-szerű tulajdonságokat úgy vezették be, hogy a vizsgálandó téglalap alakú képrészletet először diszjunkt téglalapokkal fedték le. Az egyes téglalapokra ezután kiszámolták az alattuk lévő képpontok intenzitásértékeinek összegét a fent említett konstans idejű módszerrel. Végezetül a téglalapokra kiszámolt értékeket súlyozva összegezték, ahol a súlyok lehettek -1 és +1. Mivel egy téglalapot sokféleképpen lefedhetünk diszjunkt téglalapokkal (lásd 39. ábra), ezért egy konkrét objektum detektálásához először az objektum detektálására be kell tanítani a neuronháló alapú modellt.



39. ábra. Egy lehetséges haar-like feature készlet a neuronháló betanításához.

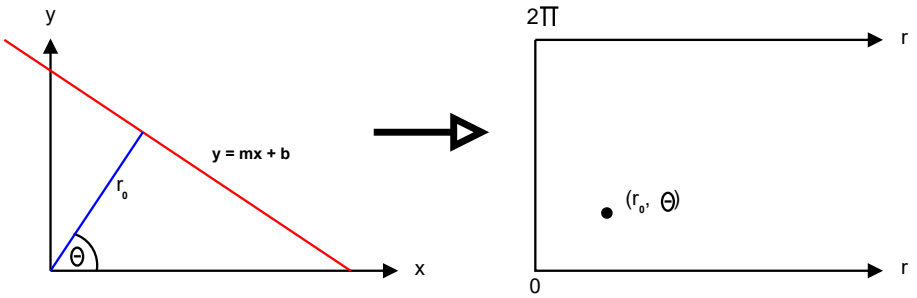
A neuronháló alapú felismerésnél három különböző képhalmazt különböztethetünk meg. Az első két képhalmaz úgy van kialakítva, hogy az elsőben az észlelendő objektum különböző előfordulási esetei találhatók, míg a másodikban olyan képek vannak, amelyeknek egyetlen képrészlete sem hasonlíthat az észlelendő objektumhoz. A két képhalmaz segítségével a tanítás során kiválasztják azokat a haar-szerű jellemzőket, amik leginkább jellemzők az objektum bizonyos képrészleteire, minden, az objektumtól eltérő kép képrészletével szemben. Az így kapott gyenge tanuló algoritmust ezután tesztelik a harmadik képhalmaz képeivel, és megvizsgálják a helyes és helytelen döntések arányát. Elsődleges cél, hogy ilyenkor minimális legyen azon rossz döntések aránya, amikor elutasítjuk a létező, észlelendő objektumot. Mivel ilyenkor az algoritmus túlságosan sok

képrészletet nyilvánít objektumnak tévesen, ezért a módszert tovább finomították. Ennek az volt a módja, hogy detektálás esetén, a már felhasznált haar-szerű tulajdonságtól eltekintve, a többi közül próbáltak meg kiválasztani egy olyat, ami tovább javította az osztályozást. Ilyen módon gyenge tanuló algoritmusokból egy robosztus felismerő módszert készítettek. Ezt a módszert vízesésnek (cascade) hívják és a tanulás meggyorsítására a jól ismert AdaBoost eljárás használható fel. A tanulás végeztével a neuronok súlyozását rögzítik, ami a detektálandó objektum tulajdonképpeni vizuális tulajdonságait rögzíti.

A Viola és Jones által bemutatott módszert előszeretettel használják az objektumdetektálás témakörében, mivel a téglalapok méretét növelve elegendő ismét végigpásztázni a képet és így különböző méretű, azonos objektumok is felismerhetőek. A módszernek egy gyorsított változatát alkalmazza az OpenCV szoftvercsomag, amely a Canny szűrőt használja fel a kép előszűrésére. Bár a módszer igen kecsegtető, a térképek speciális volta miatt néhány hátránnyal rendelkezett, melyek miatt mégsem került alkalmazásra. Hátrányai között említhető, hogy nem forgatás invariáns, nem használja ki a térképi színek csekély számát, valamint időigényes a betanítása, még úgyis, hogy pl. az OpenCV szoftver képes automatikusan torzítani az objektum képét a betanításhoz, amennyiben kevés tanulóminta áll rendelkezésre.

7.2. Szelvényvonalak Hough transzformáció alapú detektálása

Mivel a Hough transzformáció klasszikus értelemben vett változata vonalak detektálására lett kitalálva (a későbbiekben ennek számos általánosított változata született), ezért érdekes ötletnek tűnt, hogy a térképi szelvényvonalak ennek segítségével felismerhetőeke.



40. ábra. A kép xy síkjában lévő pont és a megfeleltett Hough transzformáció terében lévő pont.

A Hough transzformáció egy olyan paramétersíkon mozog (lásd 40. ábra), melynek pontjai a kép síkjának egy-egy egyenesével feleltethetőek meg. Az egyeneseket az origótól vett r távolságukkal, valamint az origóhoz legközelebb eső pontjukat az origóval összekötő egyenesnek az x tengellyel bezárt θ szögével adjuk meg. A Hough transzformáció síkjának (r, θ) pontjai és a képsík $y = mx + b$ alakban felírható egyenesei közötti

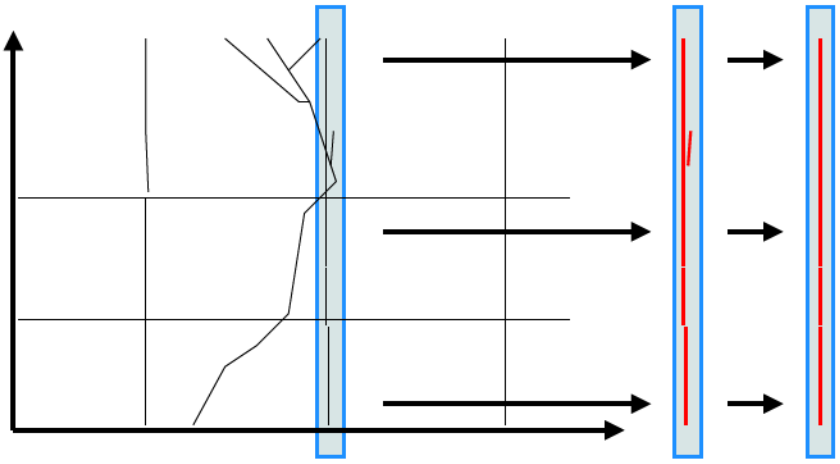
kapcsolat a következő módon adható meg:

$$r = x\cos\theta + y\sin\theta, \quad 0 \geq r \neg \mathbb{R}, \theta \neg [0, 2\pi)$$

A kép egy adott (x_0, y_0) pontján átmenő egyenes ez alapján leírható a

$$r(\theta) = x_0\cos\theta + y_0\sin\theta,$$

képlettel, ahol r értékét θ függvényében kaphatjuk meg. Ez megfelel egy (r, θ) síkon vett görbének, mely egyedi a pontra nézve. Amennyiben a kép két különböző pontjához megfeleltett görbék a Hough térben metszik egymást, úgy a képen a metszéspontnak megfeleltethető azon vonalat sikerült detektálni, amely áthalad mindkét ponton. Általánosabban megfogalmazva, a kép azon pontjai, mely egy egyenest alkotnak, a Hough térben olyan színuszgörbékre képződnek le, melyek a vonalhoz rendelt Hough térbeli pontban metszik egymást.



41. ábra. Pásztázó módszer a vektorizált és egyszerűsített töröttvonalakra épülő szelvényvonal detektálásra.

A transzformáció több problémát is felvetett az implementáció és az interpretáció kérdésében, például a Hough tér pontjainak küszöbölését, illetve azt, hogy a transzformáció nem ad információt a detektált vonalak hosszáról. Erre annál is inkább szükség lenne, mivel tudjuk, hogy a vízszintes és függőleges szelvényvonalak a térképen teljesen végigfutnak. Ez alapján tehát a térkép magasságával megegyező hosszú függőleges és a szélességével megegyező hosszú vízszintes vonalakat kellene találni. Mivel a transzformáció többlet időráfordítást jelent a feldolgozás során, ezért célszerűségi okokból a vektorizált vonalas adatokon alapuló megközelítés jobb módszernek bizonyult. Itt az elágazásnélküli maximális hosszú töröttvonalakat egy egyszerűsítésnek vetjük alá, majd irányuk alapján szelektáljuk a függőleges és vízszintes csoportba sorolhatóakat. A függőleges és vízszintes vonalakat külön csoportba tesszük, majd a csoporton belül rendezzük a vonalakat

centroidjuk koordinátája szerint sorba, azaz pl. a függőleges vonalakat x koordinátájuk szerint. A lényeg ezután az, hogy egy megadott vastagságú pásztázó vonalat viszünk végig a képen úgy, hogy segítségével a csoportjának megfelelő, általa aktuálisan lefedett szakaszok összhosszát mérjük. Az összhossz számítása a vonalaknak a pásztázó vonallal párhuzamos koordinátatengelyre vett projekciója által fedett hosszt jelenti (így az általuk vett intervallumfedéseket csupán egyszeresen számítjuk, lásd 41. ábra). Amennyiben a fentebbi módon számolt hossz kellően megközelíti a pásztázóvonal hosszát, úgy az általa vizsgált vonalakról feltételezhetjük, hogy azok egy szelvényvonalat alkotnak. A módszer finomításaként tovább vizsgálható az, hogy az átlapoló intervallumokat lefedő vonalak közül melyik illeszkedik a legjobban a szelvényvonal egyenesébe.

8. Detektált térképi objektumok vektorizálása

A térképi objektumok felismeréséhez szorosan kapcsolódik azok vektorizálása. A vektorizálás során a felismert objektumoknak megadjuk azok vektoros reprezentációját. A pontszerű objektumokat két dimenziós pontok koordinátáival ábrázolhatjuk, a vonalas objektumokat a felismert pontjaik alapján a belőlük képzett vonalszegmens- sorozatokkal. A felületeknek poligonokat feleltethetünk meg, míg a poligonokat a határukat megadó vonalszegmens-sorozattal reprezentálhatjuk (itt külön ki kell térni a lukas felületekre, illetve arra, hogy a szomszédos poligonok metszete a közös határuk legyen).

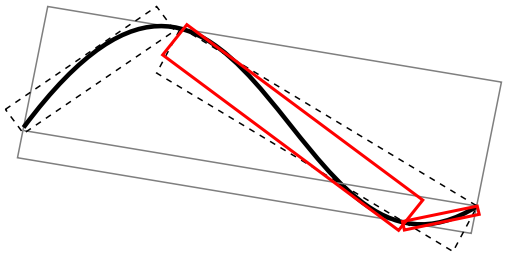
Mivel a pontszerű objektumok felismerése egyben azt is jelenti, hogy az objektumok vektoros ábrázolását megadó koordinátákat kapjuk meg, ezért itt további dolgunk nincs.

A vonalas objektum esetében a hozzá rendelt pontsorozat csupán egy töröttvonalat ad meg, amely a vonalas objektum diszkrét pontokban vett mintavételezéseként fogható fel. Ahhoz, hogy vektoros modelljét megkapjuk, a pontsorozatot egyszerűsítenünk kell, melyre a következő alfejezet mutat példákat.

A színosztolyozás és az interpretációs szabályokon alapú javítások végrehajtása után a felületeket, mint azonos színű, összefüggő pontok halmazát értelmezhetjük. A feladat tehát az, hogy ezeknek a ponthalmazoknak megkeressük a legkisebb befoglaló poligonját, amelyet a határolóvonalával adunk meg. Határoló vonal alatt a poligonnak a külső és belső határát, esetleg határait (lukas felület esetén) megadó vonalszegmens sorozatokat értjük. Ezeket a sorozatokat egyrészt ugyanúgy egyszerűsítésnek kell alávetni, mint a vonalas objektumok esetén, másrészt a szomszédos poligonok közös határvonala a poligonok metszete kell legyen. Mivel a nyers, pixel-alapú vektorizálásnál ez utóbbi feltétel nem teljesül, így egy köztes modellt kell majd a határok leképezéséhez alkalmazni. A szomszédos poligonok határvonalára vonatkozó feltétel rendkívül fontos a geometriai adatok tárolása és a térinformatikai adatbázisokban alkalmazott lekérdezések szempontjából (pl. szomszédság, metszet stb. kérdeseknél).

8.1. Vonalas jelkulcsi elemek vektorizálása

A vonalas jelkulcsi elemek felismerésénél kapott vonalszegmens-sorozatokat pontsorozatok adják meg. Egy pontsorozatot tekinthetünk egy adott görbe diszkrét mintavételezésének eredményeként is. Ekkor a feladat a görbe egyszerűsítése oly módon, hogy azt egyenes szakaszokkal közelítjük. A [18] könyv Peucker módszerét mutatja be, amelynek a segítségével a görbéket vonalszegmens- sorozatokkal közelíthetjük. Az ötletet a befoglaló téglalapok felhasználása adja. A görbéhez megadjuk kezdetben a legkisebb befoglaló téglalapját (röviden LBT), majd megvizsgáljuk ennek területét. A későbbiekben, ahogy a görbét szakaszokra bontjuk, a görbeszakaszokat lefedő téglalapok együttes területének és a görbe hosszának hányadosát szeretnénk egy előre megadott ϵ küszöbérték alá csökkenteni. Ennek jelentése az, mintha a görbét egy ϵ átlagvastagságú, azt lefedő görbével közelítenénk. A görbe felosztását iteratívan végezzük. Minden egyes iterációban kiválasztjuk azt a görbeszegmenst, amelyhez a legnagyobb területű LBT tartozik. A görbeszegmenst ott vágjuk képzeletben ketté, ahol az a végpontjait leszámítva, érintkezik a befoglaló téglalapjával (lásd 42. ábra). Az osztópont meghatározása után az LBT-ket kiszámoljuk, majd újra ellenőrízzük a közelítés jóságát. A fentebb említett ϵ küszöbértéktől függően az iterációt újrakezdjük, vagy megállunk. Megállás esetén a vonalszegmens-sorozatokat a téglalapokban lévő görbék végpontjainak sorozataként kapjuk.

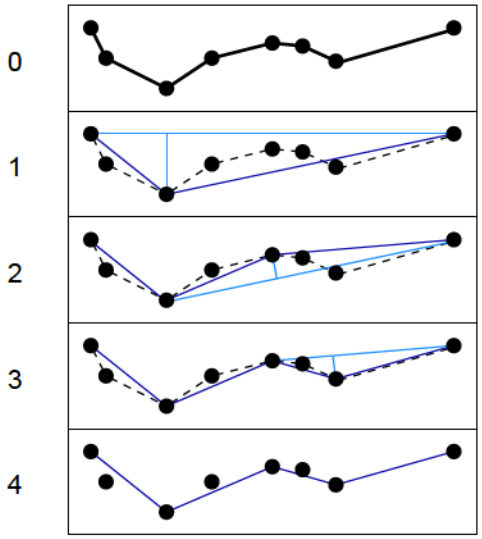


42. ábra. A görbe és annak dekompozíciója szegmensekre.

Mivel a befoglaló téglalapok kiszámítása költséges művelet, valamint a feldolgozandó görbék eleve nem folytonosak, hanem pontsorozatokkal adottak, ezért a következőkben ismertetett Douglas-Peucker algoritmus sokkal könnyebben alkalmazható.

A Douglas-Peucker által (1973-ban) adott algoritmust [8] tőlük függetlenül Ramer is felfedezte közel ugyanazon időben (1972-ben). Az algoritmust szokás az irodalomban *iteratív végpont illesztő* vagy *megoszt-és-összefésül* algoritmusként is nevezni. A kiinduló görbét mint pontok rendezett halmazát tekintjük. A közelítés hibájára ϵ mértékű toleranciát engedünk meg. A módszer szintén iteratívan osztja fel a görbe pontsorozatát, azonban az osztópontot eleve csak a megadott pontok közül választhatjuk ki (lásd 43. ábra).

Az eljárás kezdetben megjelöli a kezdő és végpontot, melyeket a közelítés mindenképpen kell, hogy tartalmazzon. A kezdő és végpontot összekötő szakasztól legtávolabbi pontot kiválasztjuk, majd megvizsgáljuk, hogy az ϵ-nál távolabb van-e a szakasztól vagy közelebb. Amennyiben közelebb, úgy valamennyi a szakasz végpontjai eső pontot elhagyjuk. Ha a kiválasztott pont távolabb van a szakasztól, mint ϵ, akkor ez a pont a



43. ábra. Példa a görbét reprezentáló pontsorozat egyszerűsítésére.

közelítés része marad, és a pontsorozatot két részre osztja. Ezt a felosztási módszert rekurzívan végrehajtjuk a kapott két szakaszon, amíg megkapjuk a pontsorozattal adott görbe ϵ-tól függő közelítését. Az alábbi pszeudokód az algoritmus működését mutatja be:

Douglas-Peucker(PontLista, ϵ)

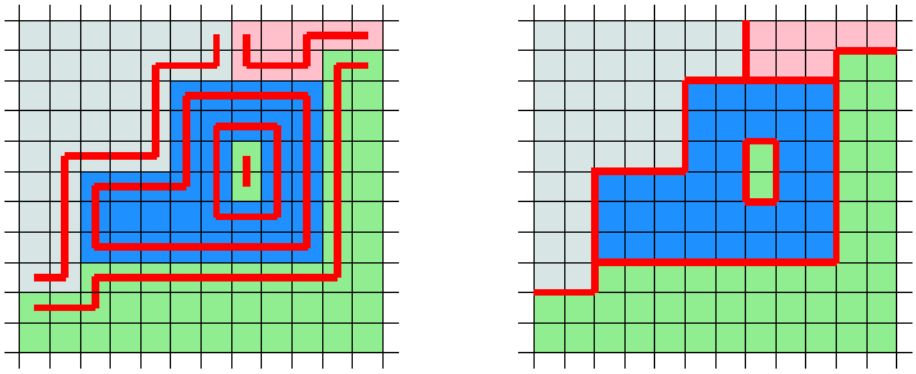
1. *dmax* ↑ 0
2. *index* ↑ 0
3. **for** *i* ↑ 2 **to** (Hossz(*PontLista*) - 1)
4. *d* ↑ P-E-Távolság(*PontLista*[*i*], Szakasz(*PontLista*[1], *PontLista[utolsó]*))
5. **if** *d* > *dmax*
6. *index* ↑ *i*
7. *dmax* ↑ *d*
8. **if** *dmax* ~ ϵ
9. *RekEredmeny*1 ↑ DouglasPeucker(*PontLista*[1..*index*], ϵ)
10. *RekEredmeny*2 ↑ DouglasPeucker(*PontLista*[*index*..*utolsó*], ϵ)
11. *EredményLista* ↑ }*RekEredmeny*1[1..*utolsó* 1].*RekEredmeny*2[1..*utolsó*]|
12. **else**

13. *EredményLista* ↑ }*PontLista*[1], *PontLista*[*utolsó*]|

14. **return** *EredményLista*

8.2. Felületek (poligonok) vektorizálása

A raszteres felületi rétegek előállítása után egy olyan kép áll rendelkezésünkre, ahol a pixelek színindexe egyértelműen azonosítja, a pixel mely felülettípushoz tartozik. Ezek ismeretében az összefüggő színindexszel rendelkező pixelek alkotta területeket (poligonokat) kell vektorizálni. Ez megtehető úgy, ha bejárjuk a homogén területek határait. Egy terület határának értelmezése többféle módon is történhet. Tekinthetjük például azt az esetet, amikor morfológiai szűrő segítségével töröljük az objektumok belső pontjait, majd pedig bejárjuk a megmaradt határoló pontokat. Ez a megközelítés az objektumok között egy pixelnyi hézagokat hagy (lásd 44. ábra).



44. ábra. A poligon töröttvonalas határának vektorizálása.

Egy másik módszer, amikor egy képzeletbeli rácshálót készítünk úgy, hogy a rácsvonalak a raszteres kép pixelei között futnak. A rácshálót ekkor egy olyan gráfnak tekintjük, ahol a csúcsok a metszéspontokkal, az élek pedig a metszéspontokat összekötő szakaszokkal vannak megfeleltetve. Azon metszéspontok lesznek valamely poligon határához tartozóak, amelyeknek szomszédai között legalább két eltérű színindexű található (lásd 44. ábra). A kép adott pontja alatt található poligon az alábbi főbb lépésekkel vektorizálhatjuk:

1. A megadott pontból elárasztásos módszert alkalmazva legyűjtjük a vektorizálandó poligonhoz tartozó pixelek koordinátáit. Az eljárás során a koordináták közül csak azokat tartjuk meg, amelyek poligon határpontjai (ebbe az esetleges lukak határpontjai is beleértendőek).

2. A határolópontokat besoroljuk a poligon egyes határaihoz oly módon, hogy amíg a határolópontok halmaza nem üres, addig abból egy találomra kivett pontból bejárást indítunk. A bejárás során a bejárt pontokat töröljük a határolópontok halmazából, és az aktuális poligonhatárhoz hozzávesszük.

3. A poligon külső határának pontjait rácsra illesztjük.

4. A poligon egyes lukait körülvevő határolópontok alapján megkeressük azokat a poligonokat, amelyek a kérdéses lukakat képezik. A lukakhoz ezután elegendő a megtalált poligonokra hivatkoznunk.

5. A poligon külső határát egyszerűsítjük a Douglas-Peucker eljárással.

Egy poligonhatárnak egy halmazban tárolt pontjait a következő módon illeszthetjük a rácsra. Kiválasztjuk a halmaz azon pontját, melynek koordinátája a kép bal felső sarkához legközelebb esik. A halmazban lévő pontkoordinátákat lexikografikus rendezés mellett keresőfában nyilvántartva ez egyetlen konstans művelettel megoldható. A kapott kezdőponttól órajárással megegyező irányban haladunk végig ezután a poligon határán. A rákövetkező pontot úgy keressük, hogy a pontnak a vele azonos színű É, D, K, NY irányokban lévő szomszédait órajárással megegyező bejárási sorrendben keressük a képen. Ha létezik a pontnak ilyen szomszédja, akkor megszámoljuk ezen szomszédnak a vele azonos színű szomszédjainak számát. Ha ugyanilyan színű minden szomszédja (belső pont), vagy éppen egyetlen sem ilyen színű (izolált pont), akkor a pont szomszédját elvetjük és a következő szomszédot vesszük sorra, egyébként megtaláltuk a rákövetkező pontot. Ezt a bejárást addig folytatjuk, amíg vissza nem érünk a kezdőpontba.

A fenti eljárás egész értékű koordinátáit úgy kell tekintenünk, hogy azok fél pixellel vannak eltolva a valóságban dél-kelet irányban. Szomszédság alatt az eljárás során mindvégig a 4-szomszédság értendő. A fenti eljárással kapcsolatban a rákövetkező pont keresése közben különböző esetek fordulhatnak elő: a lehetséges szomszéd előállított koordinátája a kép belsejében van, vagy a kép szélén fekszik, vagy pedig a képen kívülre esik. A kép belsejében lévő pontra a fenti eljárás tökéletesen működik, míg a képen kívülre eső koordináta esetén egyszerűen tovább kell keresni egy újabb szomszédot. A kép határán fekvő esetleges rákövetkező pont vizsgálatánál arra kell figyelni, hogy a kép szélén nem léphetünk órajárással ellentétes irányban, így például a kép felső határán nem léphetünk balra stb.

Látható, hogy a módszer a határpontok külső szomszédjain lépked végig, miközben a képen marad. Amennyiben a poligonhoz tartozó pixelek száma N, úgy az elárasztás futási ideje $T_1(N) = N$. Az elárasztáskor vizsgált tulajdonság, hogy a pont a poligon határán fekszik-e, 4 műveletet igényel, hiszen 4 szomszéd színét kell vizsgálni. A kapott határpontokon ezek után minden pontnak 4 szomszédját vizsgáljuk meg legfeljebb. Ebből adódik, hogy n határpont esetén ez $T_2(n) = \theta(n)$ műveletet igényel, legrosszabb esetben pedig $4n$-et. Összességében az algoritmus legrosszabb esetben $5N + 4n$ lépésszámmal lefut, ami lineárisnak tekinthető a poligon területéhez viszonyítva.

9. ShapeFile adattárolási formátum

A térinformatikai adatok feldolgozásakor gyakran több szoftverrel együttesen lehet csak megvalósítani az összes szükséges műveletet, így biztosítani kell közöttük az adatok hordozhatóságát. Az ipar egyik legelterjedtebb formátuma az ESRI által kifejlesztett

ShapeFile formátum. A szabvány pontok, vonalszegmens-sorozatok és poligonok tárolását teszi lehetővé, azonban egy vektoros állományon belül a geometriai típusok nem keverhetőek. A három alaptípus közül néhánynak rendelkezésre áll 2- és 3-dimenziós, valamint mértékkel rendelkező változata is. A felsoroltakon kívül lehetséges még pl. ponthalmazok tárolása is. A ShapeFile formátum vektoros állományokként három fájlból áll, ezek közül egy tárolja a vektoros adatokat (.SHP), míg a második tárolja az első fájl adatainak indexelését (.SHX), és végül a harmadik egy egyszerűsített táblaszerkezetben dBase formátumban (.DBF) tartalmazza az egyes objektumokhoz tartozó további nem geometriai attribútumokat. A formátum által a következő adattípusok tárolása támogatott:

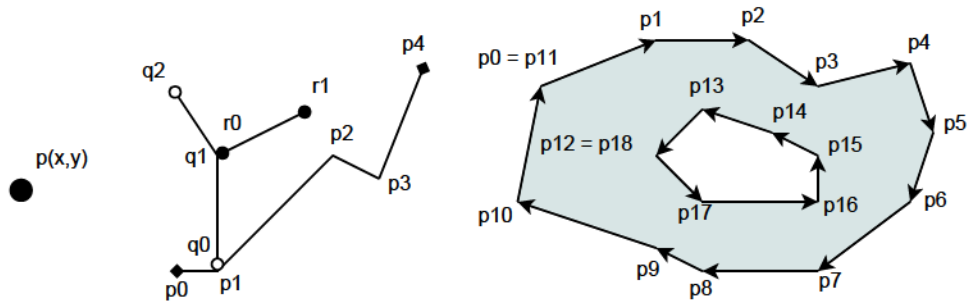
- Üres helyörző alakzat
- 2/3D-s, Mértékkel rendelkező Pont
- 2/3D-s, Mértékkel rendelkező Vonalszegmens-sorozat
- 2/3D-s, Mértékkel rendelkező Poligon
- 2/3D-s, Mértékkel rendelkező Ponthalmaz
- MultiPatch (háromszögsorozattal adott).

A formátum technikai specifikációja szabadon letölthető az internetről [10].

10. Geometriai adatok reprezentációja és tárolása

A korábbi fejezetekben történt vektorizálások eredményeként három különböző típusú vektoros adat jöhetett létre: pont, vonalszegmens-sorozat és poligon. A pontot a koordinátájával ábrázolhatjuk, a vonalszegmens-sorozatatot pedig a csatlakozási- és végpontjainak koordináta sorozatával. A vonalszegmens-sorozat önmagában nem alkalmas olyan vonalas adatok leírására, melyek elágazásokat tartalmaznak, ezért azokat több sorozattal kell leírni. A poligon határait, határonként egy-egy vonalszegmens-sorozattal adhatjuk meg oly módon, hogy a sorozat kezdő és végpontja meg kell egyezzen, továbbá külső határának a pontjait órajárással megegyező irányban, míg belső határainak pontjait órajárással ellentétes irányban kell felsorolni (lásd 45. ábra). Mivel a kezdő és végpontok meg kell egyezzenek a határoknál, ezért egyszerűségi okokból szokás tároláskor a poligon határait egyetlen sorozattá konkatenálni.

Az adatok ábrázolásának építőkövét, a pontot, a téradatbázisok különféle módokon tárolhatják a beszúrás, törlés, módosítás és keresés műveletekre való optimalizálásból kifolyólag (lásd 46. ábra). Az adatszerkezetekkel kapcsolatban bőséges információ áll rendelkezésre a [18] könyvben, így ezek közül csak néhány kerül most bemutatásra.



45. ábra. Pont, vonalszegmens-sorozat és poligon reprezentációja.

10.1. Szekvenciális lista, invertált lista

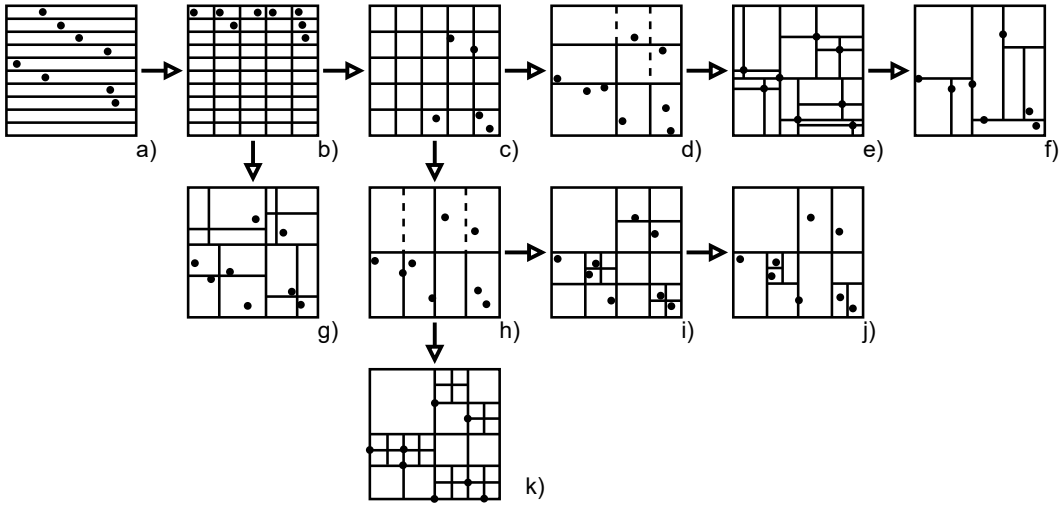
A *szekvenciális lista* a legegyszerűbb adatszerkezetnek tekinthető, ahol a pontokat leíró k attribútumot tartalmazó rekordok egymás után kerülnek felsorolásra. Ebben a szerkezetben ezért a keresés n pont esetén $n \circ k$ ideig tart, mivel minden attribútumot meg kell vizsgálnunk. Az *invertált listában* való keresés valamivel hatékonyabbnak tekinthető, mivel itt az adatok két listában vannak tárolva, az elsőben x, a másodikban pedig y koordináta szerint rendezve.

10.2. Grid index, Grid fájl

A *Grid index* lényege, hogy a teret a koordináta tengelyek valamelyikével párhuzamos egyenesek halmaza osztja fel. Két dimenziós esetben ez azt jelenti, hogy az adatok téglalapokba kerülnek. Az egymás feletti téglalapoknak a szélessége, míg az egymás melletti téglalapoknak a magassága kell megegyezzen. Speciális esetben megegyezhet valamennyi téglalap mérete, azonban az adatbázisműveletek hatékonysága miatt célszerű a vízszintes és függőleges osztóvonalak eloszlását a térben az adatokhoz igazítani (így legfeljebb adott számú adat kerül egy téglalapba). Az osztóvonalak meghatározására több ismert eljárás is létezik. Fontos megjegyezni, hogy az egy téglalapba kerülő adatok listában tárolódnak, így ezen a szinten a keresés már lineáris időben történik. A *Grid fájl* lényege egy előre rögzített térfelosztással rendelkező Grid szerkezet tárolása és karbantartása a háttértárolón. Ennek oka, hogy a háttértárolón a téglalapok külön helyeken vannak tárolva és a téglalapok felosztása, azaz a tárterület újraszerveze költséges lenne.

10.3. EXCELL

Az *EXCELL módszer* a fa adatszerkezetek közé tartozik. Ez egy bináris fa és egy könyvtár együttese, ahol a fa egy tömbben kerül tárolásra. A reprezentáció segítségével a tömbben binárisan lehet keresni az első koordináta alapján, így az adatokat a címek kiszámításával érthetjük el. A pontok további attribútumai a könyvtárakban érhetőek el. Több dimenziós esetre a módszer általánosítása egy EXHASH nevű hasítófüggvénnyel



46. ábra. Pont adatokat tároló főbb térinformatikai adatszerkezetek hierarchiája (Hanan Samet könyvének ábrája alapján): a) Szekvenciális lista, b) Invertált lista, c) Grid index, d) Grid fájl, e) Pont négyfa, f) K-D fa, g) Dinamikusan kvantált piramis, h) EXCELL, i) PR négyfa, j) PR K-D fa, k) MX négyfa

történik oly módon, hogy a koordináták legnagyobb helyiértékű bitjeit fűzi össze, majd ez alapján megkeresi ezt a helyet binárisan a tömbben. Az EXCELL a Grid fájlhoz hasonlóan szintén reguláris dekompozícióra épül, és grid könyvtárakat használ.

10.4. Pont négyfa, Pont K-d fa

A *pont négyfa* két dimenziós pontok, illetve általánosítás segítségével komplexebb geometriai alakzatok tárolására is alkalmas fa adatszerkezet. Alapötlete, hogy a kezdő teret egy négyzetként értelmezzük, és csak ezen belül tárolhatunk adatokat. Kezdetben a négyzetbe beillesztjük az első pontot, majd a második és további pontok beillesztésénél a következő módon járunk el. Megvizsgáljuk, hogy az új pont melyik térnegyedbe esik, illetve ott található-e már pont. Ha található másik pont a térnegyedben, akkor ezt a negyedet tovább negyedeljük. Ezt a felosztást addig folytatjuk azzal a térnegyeddel, amelyikbe az új pont tartozik, ameddig az új ponton kívül más pontot már nem tartalmaz a tér ezen része. A legfelső szinten a térnegyedeket égtájak nevével vagy számokkal címkézhetjük: ÉK/1, ÉNY/2, DK/3, DNY/4. A pontok térbeli helye egyértelműen meghatározható a címkékből képezett címkesorozatok segítségével, például: 1-1-3-2 vagy ÉK-ÉK-DK-ÉNY. A gyakorlatban csak egy megadott szintig érdemes végezni a térfelosztást, hiszen a térrészek tartalma külön tárolókban vannak elhelyezve. A *K-d fa* az adatszerkezet egy d-dimenzióra történő általánosítása, általában $d = 3$ dimenzióra, ami a nyolcfák segítségével írható le. A felosztás többféle módon történhet, például lehetnek az azonos szinten lévő térnegyedek méretei megegyezőek, azaz négyzetek. Overmars és van Leeuwen [17] megmutatta, hogy a térfelosztásnál létezik olyan partícionálási pont, hogy

tetszőleges N darab k-dimenziós pont esetén a pszeudo négyfa mélysége $]\log_{k+1} N\{$, míg felépítése $O(N \circ \log_{k+1} N)$ idejű.

10.5. Prioritásos keresőfa

A *prioritásos keresőfa* esetében feltesszük, hogy nincs két pont, melynek koordinátája megegyezhetne az adatbázisban. Ezután x szerint rendezzük a pontokat, majd egy kiegyensúlyozott bináris keresőfában tároljuk el őket az x koordináta értéküket kulcsként használva. Belső csúcsok esetében részfájuk azon pontja van hozzájuk rendelve, melynek maximális az y koordinátája úgy, hogy nem fordul elő a részfában. Ha ilyen nincs, akkor a csúcs üresen marad. Ebben az adatszerkezetben N pont tárolásához $O(N)$ helyre van szükség. Mivel a módszer csak x szerint rendezi a pontokat, ezért ez megnövelheti a keresés idejét, azonban kiválóan alkalmas semi-infinite lekérdezések végrehajtására, mint például: $[L_x : R_x]$, vagy $[L_x : \in]$, melyeket $O(\log_2 N + F)$ időben el lehet végezni.

10.6. MX, PR fa, PR K-d fa

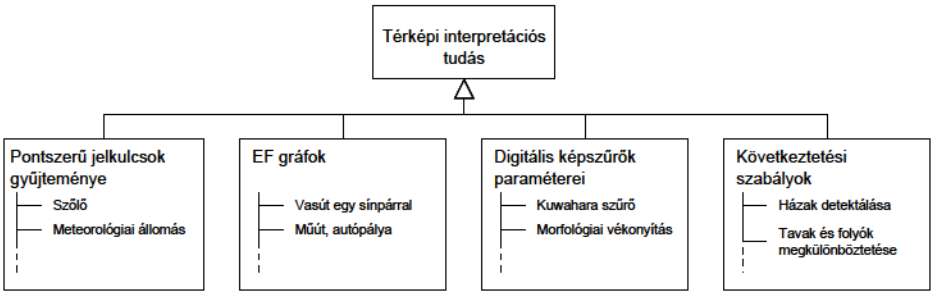
Az *MX* egy négyzetes mátrixot használ fel a pontok tárolására, ahol a pontok helyén nem nulla elem található. Ez az adatszerkezet hasonlít a tartomány négyfához, azzal a kivétellel, hogy a levelek feketék vagy üresek (van-e adat a mátrix adott pozícióján). Az adatpont itt 1x1-es négyzetként reprezentálható, míg koordinátáját a mátrix eleménak indexe adja meg, a csúcs leírását pedig egy-egy rekordok rögzíti.

Mivel az MX szerkezet csak diszkrét pontokat és véges módon képes ábrázolni, ezért a *PR négyfa* egy más megközelítést alkalmaz, amely hasonlít a tartomány négyfákéhoz, azonban a levél itt vagy üres, vagy pedig adat pontot tartalmaz, annak koordinátájával együtt. Fontos jellemző, hogy egy kvadráns csak egyetlen adatpontot tartalmazhat. A PR négyfa több dimenzióra történő kiterjesztését jelenti PR K-d fa.

Noha a fent tárgyalt adatszerkezetek közvetlenül csak pontadatok tárolására alkalmasak, azonban a gyakorlatban leginkább használtaknak léteznek egyéb geometriai elemeket tárolni képes változata is, amelyekben például vonalszegmens sorozatokat az őket meghatározó pontok beszúrásával tárolhatjuk el. A geometriai objektumok közül a poligonoknak egyik lehetséges tárolási módja, hogy a geometria alakzatnak meghatározzák a befoglaló téglalapját, majd annak sarokpontjait tárolják az adatszerkezetben, míg magát az objektumot pedig az adatbázis egy elkülönített részén. Másik lehetséges megoldás, hogy ha a tér mindazon részeiben elhelyezünk egy-egy linket az objektumra, amellyel az objektumnak nem üres a metszete. A felsorolt adatszerkezetek a térinformatikai adatbázisok működésének szempontjából kulcsfontosságúak, hiszen a jó megválasztásukon múlik az adatbázisműveletek hatékonysága (beszúrás, törlés, módosítás, keresés).

11. A térkép-interpretációs tudás ábrázolása

A 6. fejezetben megkülönböztetésre került a térkép-interpretációs tudás két szintje (lásd 30. ábra). Az első szinten szerepel a térkép vektorizálásával kapcsolatos tudás, míg a második szinten pedig a vektoros adatmodellekből további információkat kinyerő alkalmazások tudása. A vektorizálással kapcsolatban felmerül az igény, hogy az alkalmazott tudást tároljuk. A bemutatott eljárás alapján a tudáselemeknek a következő típusai különböztethetőek meg: szűrő algoritmusok paraméterezése, pontszerű jelkulcsok képeinek gyűjteménye, vonalas jelkulcsok *EF* gráfjainak gyűjteménye, felületek anomáliájára vonatkozó következtetési szabályok leírása. A tudáselemek típusainak hierarchiáját az alábbi kép (lásd 47. ábra) mutatja be.



47. ábra. A térkép vektorizálásával kapcsolatos interpretációs tudás építőkövei.

A tudás ábrázolásához célszerűségi okból az XML kínálkozott egyszerű megoldásnak, melyben a tudás elemeket a következő módon szervezhetjük:

≤ A pontszerű jelkulcsok raszteres képek, így helyettük az elérési útjukat tároljuk.

≤ Egy *EF* gráfnak egy XML csúcsot feleltethetünk meg, melynek két gyereke lesz. Az első gyerekben a gráf csúcsai fognak szerepelni, leírva a csúcsok attribútumait, míg a gráf második gyerekében a gráf élei kerülnek felsorolásra. Egy élet egy csúcs testesít meg, melynek két gyereke az éllel összekötött két gráfcsúcsra hivatkozik, és további attribútumokat adhat meg, pl. élsúly.

≤ Egy digitális képszűrőhöz egy olyan XML csúcs tartozik, amelyben név-érték módon vannak felsorolva a paraméterek.

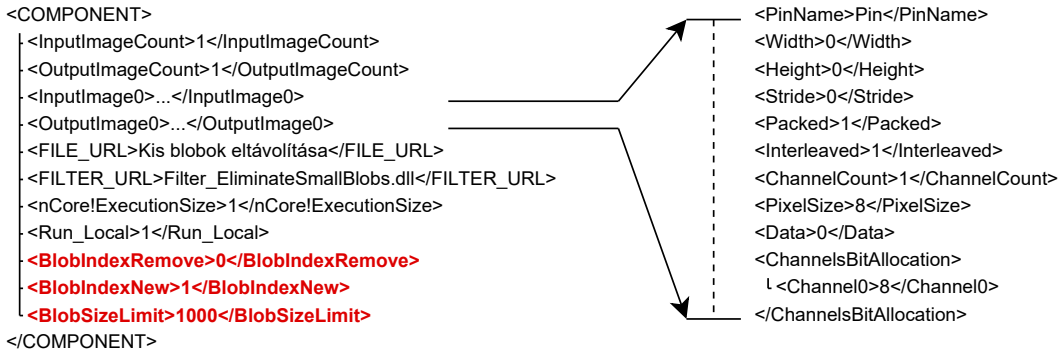
≤ Az egyes következtetési szabályok leírása a digitális képszűrők paramétereinek tárolásával analóg módon történik, mivel a szabályok kiértékelése szűrőként került implementálásra.

A következtetési szabályok informálisan az alábbi módon (lásd 8. táblázat) adhatóak meg például:

Blob típusa	Határoló típusa	Blob mérethatár	Döntés
Nem-erdő	Erdő	*n* képpont	*Típus:=Erdő*
Erdő	Nem-erdő	*m* képpont	Eltávolít

8. táblázat. Erdő-réteghez tartozó szabályok: az első az erdőn belül található kis méretű, de más típusú felületeket erdővé nyilvánítja, a második pedig a kis méretű erdőfoltokat távolítja el.

Az informálisan megadott szabályok kiértékeléséhez szűrőkre van szükségünk, ezeket pedig XML nyelven lehet paraméterezni. A paraméterezés a fenti példához az alábbi módon adódik (lásd 48. ábra):



48. ábra. Erdő-réteghez tartozó szabály XML-ben lévő leírása.

12. Teszteredmények

A vektorizálással kapcsolatos eredményeket két szempont szerint vizsgáltam. Egyrészről, hogy a kapott vektoros adatok mennyire írják le pontosan a térképet, másrészt milyen hatékonysággal bír az adott eljárás.

12.1. Hatékonyság elemzés

A hatékonyság elemzés előkészítésének első lépéseként különböző méretű, változatos tartalmú, de azonos felbontású teszttérképeket választottam ki azért, hogy kiderüljön, hogy az egyes munkafolyamatok futási idejei mennyire függetlenek a térképi tartalomtól, illetve mennyire mondhatók lineárisnak a raszteres térképek pixelszámát illetően. Második lépésként garantálni kellett, hogy a munkafolyamatok tesztjei azonos körülmények között történjenek. Ehhez egy 2GB-os permanens RAM meghajtót használtam, melyre az összes vektorizálással kapcsolatos állomány felkerült, így a műveletek idejét nem befolyásolhatta lemez töredezettségből adódó teljesítményvesztés. A tesztet három térképszelvényre futtattam le tíz munkafolyamattal, melyből az alábbi táblázatban (lásd 9. táblázat) szereplő mérési eredmények születtek.

Település **Térképszelvény** **Felbontás (300dpi)** **Munkafolyamat**	**Budakalász** **65-211** 8265x7066 ≪58.4MP	**Kardoskút** **38-334** 7910x5705 ≪45.126MP	**Úrkút** **53-343** 10978x9572 ≪105MP
`blacks`	81.952s	58.360s	120.882s
`isolines`	18.348s	14.140s	33.965s
`isolines`	25.820s	19.882s	48.470s
`points`	39.488s	22.481s	56.844s
`forests`	124.717s	94.915s	224.007s
`grasses`	152.415s	111.406s	285.138s
`houses`	54.123s	31.511s	79.002s
`lakes`	84.448s	66.931s	177.269s
`rivers`	69.048s	54.248s	128.506s
`vineyards`	244.786s	192.415s	214.410s

9. táblázat. Különböző munkafolyamatok futási idejei eltérő méretű és információmennyiségű térképek esetében.

Az összehasonlíthatóság érdekében a következő táblázatban (lásd 10. táblázat) az egyes futási idők a térképek pixelszámával kerültek normalizálásra. Feltüntetésre került ezen felül a kapott értékek munkafolyamatokra vett variánciája. Az adatokat elemezve látható, hogy az 53-343-as térképszelvény szőlővel borított felszínét feldolgozó `vineyards_SHP` munkafolyamat relatíve kétszer gyorsabban futott le, mint a többi térkép esetében. Ennek oka, hogy ez a térképszelvény nem tartalmazott szőlő felületet. A futási időkben lévő linearitástól való kisebb eltéréseket leginkább az a tény magyarázza, hogy az alkalmazott digitális szűrők működési elvei miatt nem mindegyik párhuzamosítható optimálisan a térképek vízszintes és függőleges felbontásának függvényében. Az 53-343-as térképszelvény anomáliájával és az imént tárgyaltakkal együtt is jól látható azonban, hogy a variancia igen csekély a munkafolyamatoknál, így azok futási idejei lineárisnak tekinthetők a feldolgozott pixelek számával összevetve.

12.2. Vektoros adatok minőségének elemzése

A létrejött vektoros adatmodell minőségét érdemes kategóriánként vizsgálni.

A pontszerű jelkulcsi elemek felismerésével kapcsolatban annyi vizsgálható, hogy a térképen lévő adott jelkulcsi elem típusának n példányából hányat sikerült felismerni, illetve hány pozitív és negatív tévesztés történt (téves felismerés, vagy egy előfordulás észre nem vétele). Az alábbi táblázat (lásd 11. táblázat) egy véletlen módon kiválasztott térképszelvényen történt pontszerű jelkulcsi elemek felismerésének statisztikáját mutatja be.

A pontszerű jelkulcsi elemek előfordulásainak ellenőrzését manuálisan, azaz szem-

Térképszelvény Felbontás (300dpi)	**65-211** ≪58.4MP	**38-334** ≪45.126MP	**53-343** ≪105MP	
Munkafolyamat	(T*10^6)/P	(T*10^6)/P	(T*10^6)/P	Variancia
`blacks`	1.403	1.293	1.150	0.01608
`isolines`	0.314	0.313	0.323	0.00003
`isolines`	0.442	0.441	0.461	0.00013
`points`	0.676	0.498	0.541	0.00863
`forests`	2.135	2.103	2.132	0.00031
`grasses`	2.610	2.469	2.713	0.01509
`houses`	0.927	0.698	0.752	0.01427
`lakes`	1.446	1.483	1.687	0.01682
`rivers`	1.182	1.202	1.223	0.00041
`vineyards`	4.192	4.264	2.040	1.59604

10. táblázat. A munkafolyamatok futási idejeinek a térkép pixelszámával normalizált értékei, és varianciájuk.

Jelkulcsi elem	$\sum$		H^+	H	Pontosság
Lombos erdő	5	5	0	0	100.00%
Fasor	548	535	5	13	96.72%
Facsoport	6	6	0	0	100.00%
Füves terület	86	82	4	4	90.69%
Szőlő	13	11	0	2	84.62%

11. táblázat. Pontszerű jelkulcsi elemek detektálásának statisztikája.

revételezéssel lehetett elvégezni, míg a vonalas és felületi objektumoknál ez már nem adott volna objektív adatokat, így azoknál az automatikus raszter-vektor konverzió eredményét a manuális vektorizálás eredményével lehet összevetni. Ahogy a táblázatok is mutatják, a térképszelvények mérete óriási a kézi újrarajzoláshoz, ezért közülük a legkisebb felbontásút, azaz Kardoskút térképének felületeit vektorizáltam manuálisan. Mivel ez önmagában több napi munkát jelentett, így a vonalas adatok újrarajzolásáról le kellett mondanom, hiszen ahhoz ennél is több időre lett volna szükség. A következő táblázat Kardoskút felületeinek automatikus vektorizálásából származó eredményét tartalmazza (lásd 12. táblázat).

Munkafolyamat	Egyezés	%	Fals +	%	Fals –	%
`forests`	195 637	95.86%	18 287	8.96%	8 443	4.14%
`grasses`	1 305 981	96.08%	130 532	9.60%	53 271	3.92%
`houses`	99 815	64.78%	15 785	10.24%	54 267	35.22%
`lakes + rivers`	62 158	76.44%	6 687	8.22%	19 155	23.56%
`vineyards`	514	62.61%	3 248	395.62%	307	37.39%
Szántó, más felületek kizárásával	42 895 424	99.71%	398 463	0.93%	126 418	0.29%

12. táblázat. Kardoskút 38-334 szelvényszámú térképének automatikus és manuális felület vektorizálásainak összehasonlító táblázata.

13. Konklúzió

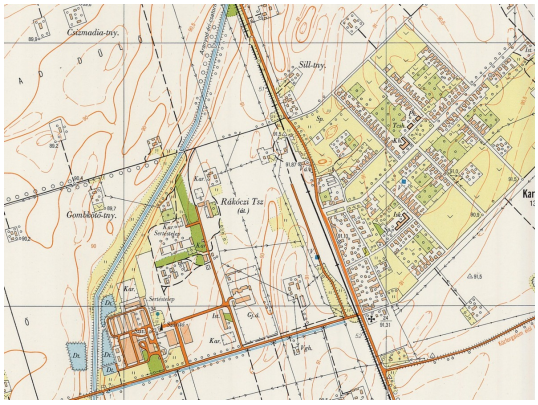
Disszertációmban a magyar topográfiai térképek raszter-vektor konverziójának automatizálási lehetőségeit mutattam be. A konverziós folyamat részletesen taglaltam az általam kifejlesztett új módszerek felhasználására építve, végezetül pedig összefoglaltam módszereim minőségi és hatékonysági eredményeit. Dolgozatomban öt új módszert vetettem fel, melyek közül négy képezi a téziseimet, egy pedig a kutatási céljaim elérését lendítette nagyban előre. A kutatásaim során kifejlesztett módszerek gyakorlati eredményeit tükrözik az alábbi képek (lásd 49. ábra Kardoskút, valamint 50. ábra és 51. ábra Budakalász térképrészleteit illetően).

Első tézisemként a korábbi, pusztán digitális szűrőkre épülő vektorizációs koncepciót egy tudás alapú megközelítéssel kívántam felváltani. Ennek keretében a térképi rétegek felismerését a papírra kerülésük fordított sorrendjében kívántam felismerni. Az egyes rétegeknél különböző típusokat különböztettem meg (pontszerű, vonalas vagy felületi objektumokat tartalmazó). Az egyes rétegek interpretálásához szabályhalmazokat definiáltam, valamint a szabályokhoz kiértékelő algoritmusokat adtam.

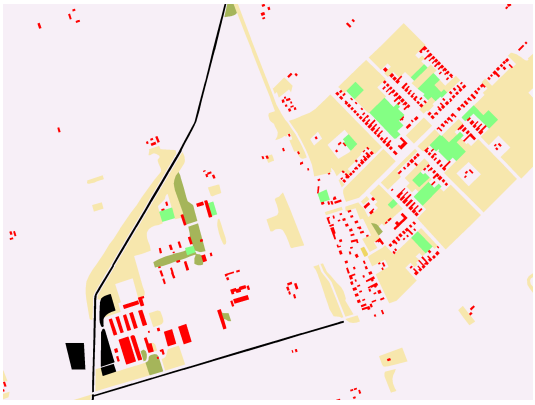
Második tézisemként a pontszerű jelkulcsi elemek felismerésére irányuló új módszeremre hivatkoznék. Ez a térképek speciális képi világát kihasználva képes a jelkulcsi elemek lineáris időben történő forgatás-invariáns felismerésére.

Harmadik tézisem a vonalas jelkulcsi elemek felismerésére létrehozott új EF gráf koncepcióját mutatja be. A gráf segítségével lehetőség van a vonalas vektoros adatokból egy a jelkulcsi elem struktúrája szerinti objektum felismerésre, legyűjtésre. A módszer előnye, hogy véges determinisztikus automatával történő megvalósítás esetén lineáris futásidővel rendelkezik.

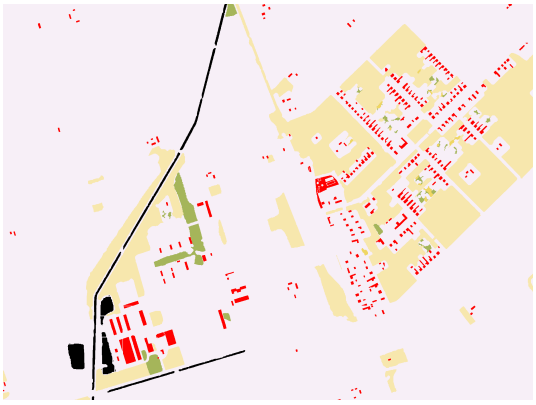
Utolsó tézisemként egy olyan új eljárást ismertettem, amely képes a legtöbb szekvenciális képszűrő algoritmus automatikus párhuzamosítására. A módszer akkor használható, ha a szűrő a memória elérését egy meghatározott sorrendben végzi, például ha



(a) Eredeti térképrészlet



(b) Manuálisan vektorizált térképrészlet



(c) Automatikusan vektorizált térképrészlet

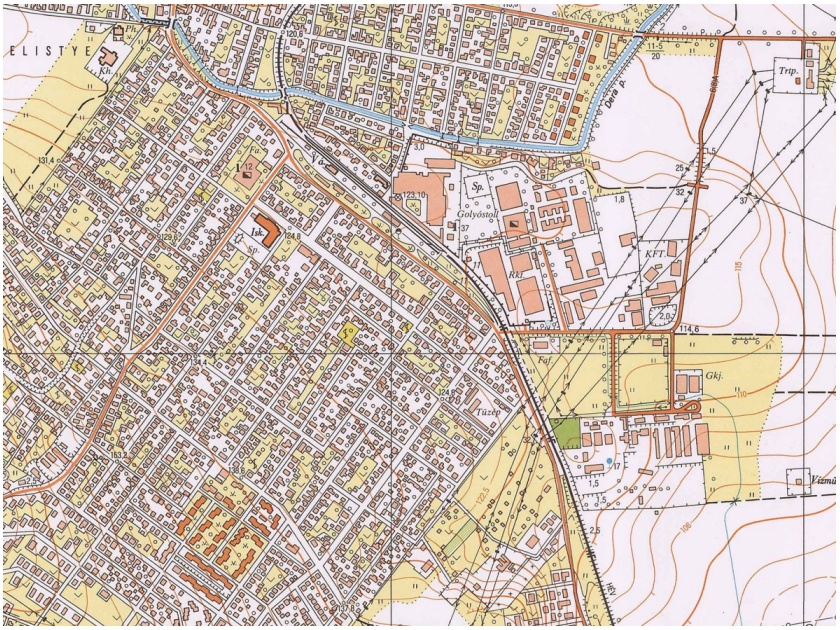
49. ábra. Kardoskút 38-334-es térképszelvényének eredetije, valamint manuálisan és automatikusan vektorizált változatai

a képet sorfolytonosan, blokkonként vagy rekurzívan dolgozza fel. Mivel a párhuzamos programozás nagyobb szellemi és időráfordítást igényel, ezért ezzel a megoldással lecsökkenthető az fejlesztési idő, továbbá nő a kód minőségi mutatója, hiszen kiküszöböli a párhuzamos programozáskor elkövethető hibákat, melyek igen nehezen deríthetőek fel.

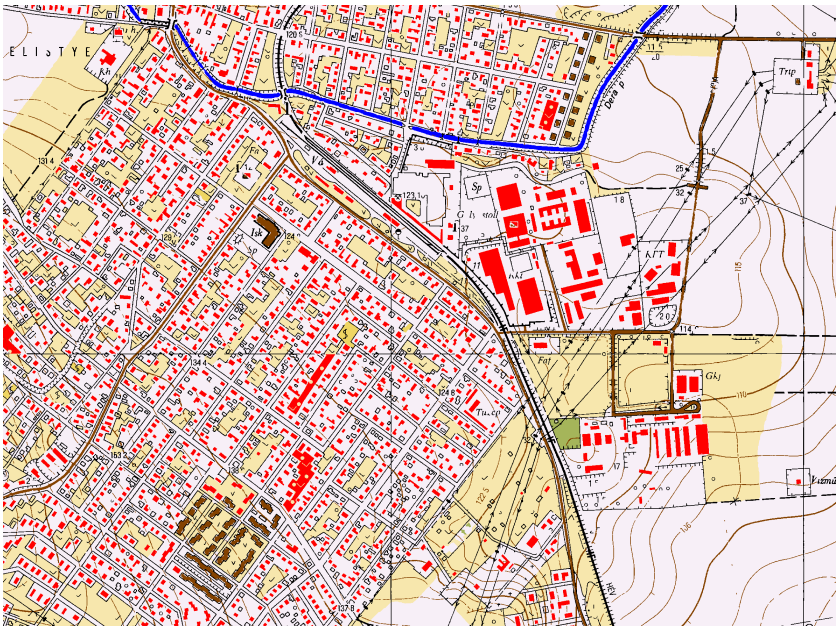
Végül, de nem utolsó sorban egy valós idejű szűrő szimulátort ismertettem, amely teljes egészében átvállalja a szűrők implementálásakor szükséges támogató kódok elkészítését. Az eszközt használva elegendő kizárólag a szűrő magját elkészíteni, nem kell foglalkozni az export, import vagy kommunikációs felületekkel. A szimulátor egy általános szűrőként viselkedik, amely képes a szűrőmagot C++ forrásnyelven befogadni, valamint a háttérben futási időben fordítani, visszaolvasni és kódbeültetéssel futtatni is anélkül, hogy ahhoz külön fejlesztőkörnyezetre lenne szükség. A szimulátor lehetőséget ad a kód véglegesítésére és ezzel az általános szűrő permanens specializációjára.

Összességében egy olyan térkép interpretációs módszert dolgoztam ki több új módszer összességeként, amely a korábbi koncepciókhoz képest jobban követi a térképolvasó

ember és a térképkészítő szakember logikáját. Az alkalmazott logika szabályrendszerben került rögzítésre, melyhez kiértékelő algoritmusokat készítettem. Mindezek mellett olyan fentebb már említett új módszerek születtek, amelyek gyakorlati haszonnal bírnak más célú képfeldolgozó eljárások esetében is (pl. automatikus párhuzamosítás).

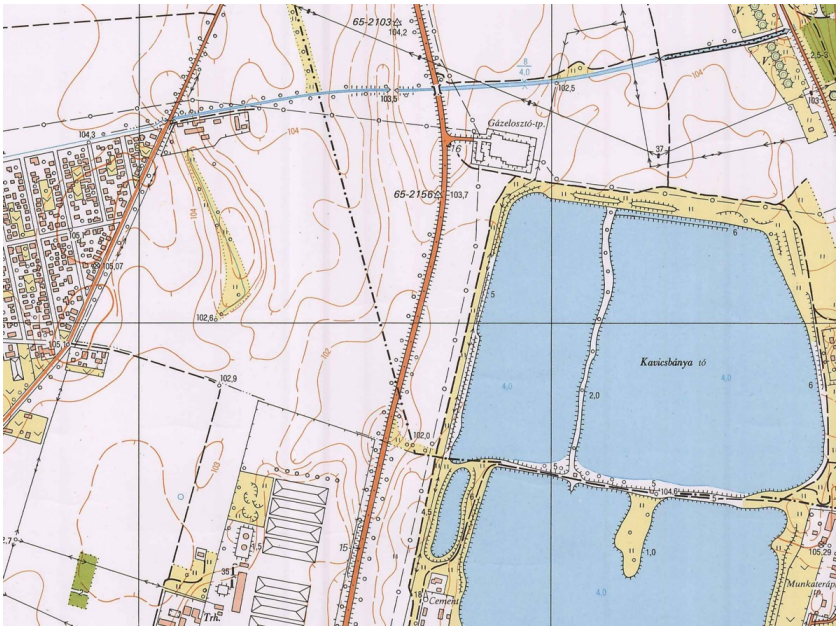


(a) Eredeti térképrészlet

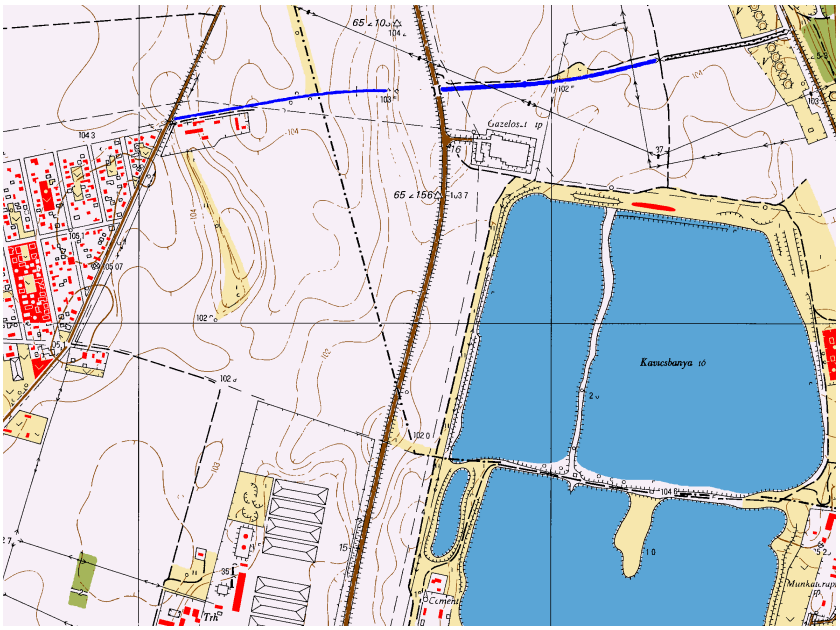


(b) Automatikusan vektorizált térképrészlet

50. ábra. Budakalász 65-211-es térképszelvény részletének eredeti és automatikusan vektorizált változata.



(a) Eredeti térképrészlet



(b) Automatikusan vektorizált térképrészlet

51. ábra. Budakalász 65-211-es térképszelvény részletének eredeti és automatikusan vektorizált változata.

Topográfiai térképek automatikus raszter-vektor konverziója

Összefoglaló

Kulcsszavak: Geoinformatika, topográfiai térképek, raszter-vektor konverzió, jelkulcs felismerés, mesterséges intelligencia, tudás ábrázolás

A papíralapú raszteres térképeket a társadalmi szintű felhasználás szerteágazó területein egyre inkább felváltják a digitális térképek. A fontosabb topográfiai térképek vektorizált, számítógépes változatait a legtöbb országban jórészt fáradtságos „kézi" eszközökkel, több év alatt készítették el. A térképek raszter-vektor konverziójának feladata azonban belátható időn belül nem szűnik meg az új térképek megjelenése és a régebbiek módosításai miatt.

A tézis az IRIS projekt kutatási eredményeire épül, ahol egy intelligens raszter-vektor konverziós rendszer és annak elméleti háttere került kidolgozásra. Célként a raszter-vektor konverzió minél magasabb szintű automatizálása került megfogalmazásra, ami a tudásalapú megközelítést helyezte előtérbe. A dolgozat a fő térképi szimbólum típusok felismerésére és konvertálására összpontosít, valamint az alkalmazott képszűrő algoritmusok párhuzamosításának automatizálására.

A pontszerű jelkulcsi elemek kis képekként azonosítanak valódi objektumokat (pl. emlékmű). A kifejlesztett eljárás a térképen található jelkulcsi elemeket a szimbólumok adatbázisában lévőkkel próbálja illeszteni azok mintázata alapján. A mintaillesztések számának csökkentésére egy élszűrés, egy Otsu küszöbölés, valamint egy további (a jelkulcsi elem pixeleinek számára vonatkozó) küszöbölő eljárás szolgál.

A vonalas jelkulcsi elemek (pl. utak, vasutak) esetében a vonalas elemek stílusát (szín, szélesség) és topológiáját is fel kell ismerni. A topológia meghatározására egy EF-gráf került kifejlesztésre, ami az F elágazási (fork) és E végpontok (end) alapján képes a jelkulcsi elem topológiáját leírni.

A felületszerű jelkulcsi elemek a felületet egy adott színnel vagy mintázattal fedik le (pl. tó, bokros terület). A homogén felületek egy tudás alapú szabály halmaz alapján kerülnek feldolgozásra, míg a textúrázott felületeket a pontszerű jelkulcsi elemekhez tartozó algoritmussal lehet azonosítani, miután a legkisebb őket generáló részletük (kernel) meghatározásra került.

Jelen munkán túlmutat néhány jelkulcsi elem (pl. szövegek) automatikus felismerése, ezért az automatikusan elkészített vektoros állomány értelemszerűen nem tartalmazza a raszteres térkép összes részletét. Megjegyzendő továbbá, hogy az algoritmusok esetleges téves detektálásból eredő hibáit szakértői javításnak kell követnie. A szakértői térképértelmezésnek fontos pontja pl. a térképi rétegek nyomtatási sorrendjének ismerete. Ezen tudás felhasználása a konverziót sokkal pontosabbá teheti.

A kutatást az ELTE Kutatóegyetemi pályázata támogatta (TÁMOP-4.2.1/B-09/1/KMR-2010-003).

Automatic raster-vector conversion of topographic maps

Summary

Keywords: Geoinformatics, topographic maps, raster-vector conversion, symbol-recognition, artificial intelligence, knowledge representation

The scanned paper-based maps in raster image format are suitable for humans, but geoinformatics prefers to use the properly converted, vectorized maps. The important topographic maps have already been vectorized in most countries by a cumbersome, manual procedure. However, the task of raster-vector conversion of paper-based maps will not become obsolete within the next few years.

The thesis is based on the researches results of the IRIS project, where the theoretical background of a raster-vector conversion system has elaborated and the prototype of the system has developed. The aim of the development is to automatize the raster-vector conversion as much as possible. This goal puts an emphasis on the knowledge-based approach. This thesis focuses on the automatic recognition and conversion of the three main types of map symbols to improve the efficiency of the recognition system and it also focuses on the automatic parallelization of raster image filters.

Point-like symbols are small icons, each representing a real object (e.g. a monument). The recognition algorithm tries to identify these symbols based on given symbol patterns. The amount of the required pattern matching is reduced by edge filtering, Otsu thresholding and an area thresholding (based on the amount of edge pixels of the symbol).

In order to identify linear symbols (e.g. roads, railroads) both line style and topology must be recognized. To determine the topology an EF graph is created using the E end- and F fork-points of the road-like graphics.

Surface-like symbols cover a region with a solid colour or with a pattern (e.g. lake or scrub). Homogenous surfaces are processed according to a knowledge-based set of rules, while texturized surfaces are identified by the algorithm used for point-like symbols after the procedure determined the smallest repetitive part (kernel) of the texture.

Currently, the automatic recognition of some kinds of map symbols (e.g. texts) is beyond the scope. Thus, the vectorized coverage generated automatically does not contain all of the elements occurring on the original raster map. Furthermore, the algorithms used for recognition provide the possibility of human expert's intervention in the case of false detection. An important point in the expertise of human interpreters is, for example, the knowledge of the order of map layers they have been printed. The inclusion of this knowledge would make the conversion much more intelligent.

The research project is supported by the TÁMOP-4.2.1/B-09/1/KMR-2010-003 project.

Hivatkozások

[1] Baik, S., et al.: A naive geography analyst system with cognitive support of imagery exploitation. Monroy, R., et al., editor, Lecture Notes in Artificial Intelligence, No. 2974, pp. 40–48, 2004.

[2] Buttenfield, B. P. and McMaster, R. B., editors. Map generalization: Making rules for knowledge representation. Longman Scientific & Technical, 1991.

[3] Chen, H., et al.: A geographic knowledge representation system for multimedia geospatial retrieval and analysis. International Journal on Digital Libraries, No. 1, Vol. 2, pp. 132–152, 1997.

[4] Corner, R.J.: Knowledge Representation in Geographic Information Systems. Ph.D. thesis, Curtin University of Technology, 1999.

[5] Dekker, A.: Kohonen neural networks for optimal colour quantization. Network: Computation in Neural Systems, Vol. 5, pp. 351–367, 1994.

[6] Dezső, B., Elek, I., Máriás, Z.: IRIS, Development of Automatized Raster-Vector Conversion System. Tech. rep., Eötvös Loránd University and IKKK, 2007.

[7] Dezső, B., Elek, I., Máriás, Zs.: Image processing methods in raster-vector conversion of topographic maps. Karras, A. D., et al., editor, Proceedings of the 2009 International Conference on Artificial Intelligence and Pattern Recognition, pp. 83–86, 2009.

[8] Douglas, D. and Peucker, T.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer, 10(2), pp. 112–122, 1973.

[9] Ebi, N. B.: Image Interpretation of Topographic Maps on a Medium Scale Via Frame-based modelling. International Conference on Image Processing, IEEE Press, California, Vol. I. pp. 250–253, 1995.

[10] ESRI: ESRI Shapefile Technical Description. An ESRI White Paper – July 1998, http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf, 1998.

[11] Janssen, R. D. T. és Vossepoel, A. M.: Adaptive vectorization of line drawing images. Computer Vision and Image Understanding, 65(1), pp. 38–56, 1997.

[12] Katona, E.: Automatikus térkép-interpretáció, Szegedi Tudományegyetem, 2000.

[13] Klinghammer, I., Papp-Váry, Á.: Földünk tükre a térkép. Gondolat, 1983.

[14] Liang, S., Chen, W.: Extraction of line feature in binary images. IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences E91A, 1890–1897, issue 8., 2008

[15] Lopes, A., Nezry, E., Touzi, R., Laur, H.: Structure detection and statistical adaptive speckle filtering in SAR images. International Journal of Remote Sensing, Vol. 14, No. 9, pp. 1735–1758, 1993.

[16] Mayer, H., Heipke, C., Maderlechner, G.: Knowledge-based interpretation of scanned large-scale maps using multi-level modeling. International Archives of Photogrammetry and Remote Sensing, 29 B3, pp. 578–585. 1992.

[17] Overmars, M. H., és van Leeuwen, J.: Dynamic multi-dimensional data structures based on quad- and k-d trees. Acta Informatica, 17(3), pp. 267–285, 1982.

[18] Samet, H.: The Design and Analysis of Spatial Data Structures. Addison - Wesley, 1990.

[19] Samet H., Soffer A.: MAGELLAN: Map Acquisiton of GEographic Labels by Legend ANalysis. International Journal on Document Analysis and Recognition, Vol. 1, pp. 89–101. 1998.

[20] Suzuki, S., Yamada, T.: MARIS: Map Recognition Input System. Pattern Recognition, Vol. 23, No. 8, pp. 919–933. 1990.

[21] Szendrei, R., Fekete, I., and Elek, I.: Texture based recognition of topographic map symbols. In Karras, A. D., et al., editor, Proceedings of the 2009 International Conference on Artificial Intelligence and Pattern Recognition, AIPR-09, pp. 7–10, ISBN: 978-1-60651-007-0, ISRST. 2009.

[22] Szendrei, R., Elek, I.: Automatic symbol recognition for topographic maps In Geomatics - Riga, pp. 42–48, 2009.

[23] Szendrei, R.: Automatic parallelization of raster image filters. In Majkic, Z., et al., editor, Proceedings of the 2010 International Conference on Artificial Intelligence and Pattern Recognition, AIPR-10, pp. 30–33, ISBN: 978-1-60651-015-5, ISRST. 2010.

[24] Szendrei, R., Elek, I., Márton, M.: Graph-based feature recognition of line-like topographic map symbols. Lecture Notes in Computer Science 6729, pp. 291-298, 2011.

[25] Szendrei, R., Elek, I., Fekete, I.: Automatic recognition of topographic map symbols based on their textures. Lecture Notes in Computer Science 6729, pp. 299-306, 2011.

[26] Szendrei, R., Fekete, I., Elek, I., Márton, M.: A knowledge-based approach to raster-vector conversion of large scale topographic maps. Acta Cybernetica 20, pp. 145–165, 2011.

[27] Tsai, D., Tsai, Y.: Rotation-invariant pattern matching with color ring-projection. Pattern Recognition, 35(1), pp. 131–141, 2002.

[28] Xin, D., Zhou, X., Zheng, H.: Contour Line Extraction from Paper-based Topographic Maps, Journal of Information and Computing Science Vol. 1, No. 5, pp. 275–283, 2006.

[29] Viola, P., Jones, M.: Rapid Object Detection using a Boosted Cascade of Simple Features. Conference on Computer Vision and Pattern Recognition, CVPR'01, Vol. 1, pp. 511, 2001.